Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

# Diseño e implementación de una aplicación web para la gestión y actividad de una tienda de ocio

**Estudante:** Adrián Gómez García
**Dirección:** Juan Raposo Santiago

A Coruña, xuño de 2021.

*A mi familia, amigos y allegados*

**Agradecimientos**

Primero para mis familiares, por demostrar incontables veces su apoyo, sin el cual no podría haber llegado hasta aquí.

A mis amigos, compañeros y personas cercanas por escucharme y empujarme a avanzar, más especificamente a Andrea, Benyi, Mauro, Pedro y Roque.

Por último, me gustaría agradecer al director del proyecto, Juan Raposo Santiago por su paciencia y asistencia a lo largo del desarrollo.

## Resumen

El objetivo de este proyecto consiste en desarrollar una aplicación web para la gestión y actividad de tiendas de ocio. La aplicación debe permitir control de las mismas tanto a los administradores de la empresa como a los dependientes de las tiendas individuales, además de ser utilizable por clientes que busquen sus servicios y catálogo. Siendo de notable interés la compra y alquiler de los productos, el alquiler de espacios y la visualización de eventos para los clientes y la gestión de dichos elementos para ambos tipos de administradores.

Los principales lenguajes que se utilizan son Java para el backend de la aplicación y Javascript para el frontend, utilizando una arquitectura por capas siguiendo el modelo de diseño cliente-servidor. El cliente se ejecutará desde un navegador, utilizando frameworks como React y Redux para su implementación. El servidor cuenta con una capa modelo que maneja la funcionalidad y acceso a base de datos además de un servicio REST, en este caso utilizando los frameworks Hibernate y Spring.

Durante el desarrollo de la aplicación se utiliza una metodología ágil, basada en iteraciones incrementales, aplicando elementos de otras metodologías como Scrum o Kanban. Con la intención de que cada incremento obtenga como resultado una versión funcional, se desarrollará en cada iteración una o múltiples funcionalidades esperadas para la aplicación.



## Abstract

The objective of this project is the development of a web application for the management and activity of a hobby store chain. This application must allow it's control to company administrators as well as shop cashiers, while remaining usable to clients looking for their services and catalog. Being of notable interest for the clients the purchase and rental of products, the rental of spaces and the display of events. Administrators must be able to manage previously mentioned elements of interest.

The main programming languages used in the project are Java in the application's backend and Javascript on it's frontend, following a layered client-server structure. The client will be run on a web browser, using frameworks as React and Redux for it's implementation. The server is formed by a model layer that manages functionality and access to the database in addition to a REST service implemented using Hibernate and Spring as frameworks.

The development will follow an agile methodology based on incremental iterations and using elements from other methodologies like Scrum or Kanban. With the intention of obtaining a functional version of the application with each iteration, one or multiple expected functionalities are developed on each iteration.

**Palabras clave:**

- Aplicación Web
- Java
- Javascript
- Hibernate
- Spring
- React
- Redux
- Metodología ágil
- Alquiler
- Venta

**Keywords:**

- Web Application
- Java
- Javascript
- Hibernate
- Spring
- React
- Redux
- Agile methodology
- Rental
- Purchase

# Índice general

# Índice de figuras

# Índice de tablas

Capítulo 1

# Introdución

## 1.1 Contexto y motivación

Se denomina tienda de ocio o *hobby store* a un establecimiento que ofrece un catálogo de productos concretos como coleccionables, juegos de mesa, juegos de rol y otros tipos de productos relacionados con la cultura *geek*. Uno de los objetivos de estos establecimientos, además de incluir y retener clientes es formar una comunidad en su contorno. Esta comunidad acude por los eventos y presentaciones de nuevos productos realizados en el establecimiento, siendo a veces un caos organizativo y con control laxo.

Estas empresas, generalmente cuentan con espacios y productos de alquiler limitados que se suelen asignar mediante reservas hechas por papel y lápiz de forma presencial o por teléfono pudiendo generar conflictos y olvidos, creando descontento entre la clientela de su establecimiento. Además, aunque estas empresas tienen presencia, están limitadas a poner su calendario de eventos de la semana en las redes sociales de forma exclusiva, no teniendo un lugar centralizado para visualizar sus eventos o catálogo disponible de forma cómoda para el usuario.

Como usuario de alguno de estos establecimientos y testigo de la falla organizativa, existe un interés personal en implementar una plataforma tanto para los administradores de la tienda de ocio como para el cliente interesado, utilizable de forma cómoda y accesible desde navegador ya sea desde un ordenador o un dispositivo móvil. Una aplicación de gestión y actividad permite a cada entidad ser más competitiva y afianzar y aumentar la comunidad y clientela, por esto la intención del proyecto es implementar una aplicación escalable, con posibilidad de añadir nuevos establecimientos y gestores de los mismos según sea necesario.

## 1.2 Estado del arte

En la actualidad, es habitual que cada empresa individual del sector tenga su propia página web personalizada, usualmente siguiendo una estructura de blog para mostrar sus lanzamientos y eventos, o en contraparte, que no exista una página web propia y utilicen exclusivamente un espacio en alguna de las redes sociales populares, haciendo que su visibilidad sea reducida al competir con otros posts y mensajes.

Mientras que la visibilidad en las redes es relevante para la formación de la comunidad que buscan, rara vez existe un espacio unificado donde poder comprobar cada uno de los elementos que debe manejar uno de estos establecimientos o empresas, haciéndolo menos competitivo, accesible o directamente una tarea ardua a la hora de realizar múltiples operaciones del negocio como puede ser el lanzamiento de un nuevo producto y su presentación asociada.

El enfoque de la aplicación desarrollada es intentar abarcar este problema, adaptándose a las necesidades de este tipo de negocio desde una perspectiva distinta a las soluciones indicadas en esta sección.

## 1.3 Objetivos

Los principales objetivos considerados a la hora de la realización del proyecto han sido los siguientes:

1. Gestión de los usuarios, pudiendo visualizar de forma sencilla su información asociada, además de poder manejar sus citas de alquiler y pedidos ya sea cancelándolos o marcándolos como pago.

2. Gestión del almacén del establecimiento o establecimientos de la supuesta empresa, siendo importante la diferenciación de los tipos de productos ofrecidos, pudiendo clasificarse como venta, venta de segunda mano o alquiler.

3. Control de los alquileres de espacios de cada establecimiento o tienda, permitiendo a los usuarios ver la disponibilidad de los mismos entre fechas concretas y a los administradores a consultar la asistencia.

4. Ampliación sencilla del número de espacios o tiendas que forman parte de la empresa, permitiendo añadir tanto establecimientos como nuevos espacios de forma sencilla.

5. Gestión básica de eventos creados por los gestores de la tienda o administradores globales para la comunidad, pudiendo ver los eventos próximos para cada establecimiento individual.

6. Expandir y afianzar conocimientos en los frameworks y lenguajes de programación seleccionados para la realización del proyecto, siendo estos conocidos con anterioridad pero con interés personal en su profundización.
7. Obtener una versión funcional de la aplicación en cada iteración, consiguiendo que cada nuevo incremento amplíe dicha versión funcional. Tratando de mantener un nivel de escalabilidad a lo largo del proyecto.

## 1.4 Estructura de la memoria

La memoria se ha estructurado de la siguiente manera:

#### 1.4.1. Introducción

Explicación breve del contexto, la motivación y los objetivos del proyecto.

#### 1.4.2. Tecnologías y herramientas

Listado y explicación de las tecnologías y herramientas utilizadas además de justificaciones sobre las mismas.

#### 1.4.3. Metodología

Explicación de las bases metodológicas y su uso en el proyecto.

#### 1.4.4. Análisis

Definición de actores, requisitos funcionales esperados de la aplicación y desgranado en historias de usuario, además de una planificación de iteraciones general.

#### 1.4.5. Diseño

Explicación de la arquitectura del proyecto generalizada junto con una explicación más detallada de las decisiones de diseño y patrones seguidos.

#### 1.4.6. Implementación

Explicación del desarrollo y elaboración de la aplicación detallando elementos de implementación del backend y frontend.

#### 1.4.7 Pruebas

Explicación del proceso de pruebas seguido durante el desarrollo de la aplicación.

### 1.4.8. Planificación y evaluación de costes

Explicación de las cuestiones de planificación en el contexto del proyecto realizado y evaluación de costes del mismo.

### 1.4.9. Conclusión

Exposición de conclusiones tras el proyecto y posibles lineas de trabajo futuras.

Capítulo 2

# Tecnologías y herramientas

Con la intención de cumplir los objetivos postulados en el capítulo anterior (sección 1.3), se ha realizado la siguiente selección de lenguajes de programación, herramientas y tecnologías utilizados en el proyecto:

## 2.1 Lenguajes de programación

### 2.1.1 Java

La implementación del backend se realiza en Java. Los motivos para su elección son la familiaridad personal en su uso, la orientación a objetos y la popularidad del mismo en el panorama actual[1].

### 2.1.2 SQL/JPQL

*Lenguaje de consulta estructurada*, más conocido por su nombre en inglés Structured Query Language (SQL), se utiliza en el proyecto a la hora de crear y rellenar con datos las tablas de la base de datos necesarias para la aplicación.

Java Persistence Query Language (JPQL) es un lenguaje de consulta orientado a objetos utilizado por la capa modelo para acceder a la información de la base de datos. JPQL cuenta con una sintaxis muy similar a SQL.

### 2.1.3 Javascript

El lenguaje principal del frontend o lado cliente. Los motivos para elegir este lenguaje son su popularidad[1], su utilidad a la hora de crear páginas dinámicas y el interés personal de profundización en el mismo, concretamente utilizando React, framework que se explicará más adelante.

### 2.1.4 HTML/JSX

Aunque no se trata estrictamente de un lenguaje, sino una extensión sobre la sintaxis de Javascript utilizada por React, JSX[2] se utiliza a la hora de crear la interfaz de usuario en los componentes del frontend. Comparte muchas similitudes con el lenguaje de marcado HyperText Markup Language (HTML) y es sencillo de leer.

### 2.1.5 CSS/Sass

Cascading Style Sheets (CSS) es un lenguaje de diseño que define el estilo de un documento escrito en un lenguaje de marcado (como HTML mencionado anteriormente). En el proyecto se utiliza el metalenguaje Sass, creando archivos de extensión .scss para manejar el diseño de la interfaz. Sass[3] es un acrónimo de Syntactically Awesome Stylesheets y actúa como azúcar sintáctico a la hora de definir los estilos de la aplicación, proporcionando ventajas entre las que destacan el uso de variables, anidamiento de bloques o los mixins que permiten reutilizar código. Estas últimas funcionalidades no se encuentran disponibles en CSS estándar.

## 2.2 Herramientas

### 2.2.1 Eclipse

Eclipse es un entorno de desarrollo integrado o Integrated Development Environment (IDE) en inglés. Este entorno de desarrollo facilita la programación del backend en Java mediante atajos de teclado útiles, funciones de auto-completado y refactorización, asistentes para la creación y depuración de proyectos y módulos o plug-ins de acceso libre que permiten integración de nuevas utilidades al entorno, entre otros beneficios.

### 2.2.2 Maven

Se trata de una herramienta de gestión de proyectos desarrollada por Apache. Maven simplifica la construcción del proyecto utilizando su Project Object Model (POM) para describir las dependencias de la aplicación y simplificando la anexión de nuevos módulos. Maven es extensible y cuenta con un repositorio de librerías.

### 2.2.3 MySQL

Como gestor de base de datos se utiliza MySQL, siendo este un sistema de gestión de bases de datos relacionales de código abierto. Su principal ventaja es la sencillez de uso e instalación.

### 2.2.4 Postman

Se trata de una herramienta de desarrollo utilizada para enviar peticiones Representational State Transfer (REST) y probar las funcionalidades exportadas mediante las Application Programming Interfaces (API) del servidor. Permite la automatización de pruebas sobre las peticiones REST.

### 2.2.5 Visual Studio Code

Es el IDE escogido para el desarrollo del frontend de la aplicación por su facilidad y comodidad de uso, como Eclipse, permite incluir plug-ins que facilitan el desarrollo y cuenta con funciones de auto-completado y refactorización, además de contar con un terminal integrado en el propio entorno de desarrollo por defecto.

### 2.2.6 Yarn

Para manejar y administrar las dependencias del frontend inicialmente se buscaba utilizar el gestor de paquetes npm ya que viene incluido en el entorno de ejecución Node.js. Por comodidad la decisión final fue utilizar Yarn como gestor de paquetes por su mayor velocidad en comparación con npm[4].

### 2.2.7 Git

Para facilitar el cumplimiento de los objetivos del proyecto, es necesario un software de control de versiones, con el cual poder mantener etiquetas con cada incremento funcional resultante de una iteración. Git tiene múltiples ventajas para el trabajo en equipo que no son aplicables a este proyecto por haber sido desarrollado individualmente.

## 2.3 Librerías y otros frameworks

### 2.3.1 JPA/Hibernate

Hibernate es un mapeador objeto/relacional de código libre que implementa y expande la definición estándar de Java Persistence API (JPA), permitiendo manejar la información contenida en la base de datos y simplificando el desarrollo del modelo de la aplicación.

### 2.3.2 Spring

La elección del framework se basa en que Spring Boot permite la creación sencilla de proyectos basados en Spring reduciendo el tiempo de configuración del mismo. El entorno de Spring permite mediante sus módulos implementar DAOs de forma sencilla y simplifica la

configuración de los controladores web de la capa REST, además de ofrecer soporte para la gestión de transacciones otorgando transparencia sobre ellas al desarrollador.

### 2.3.3 React

La librería de código abierto React fue desarrollada y es mantenida por Facebook con la intención de simplificar la creación de interfaces de usuario en Javascript. React fomenta el desarrollo basado en componentes, facilitando la reutilización de los mismos y ahorrando tiempo de desarrollo. Desde React 16.8, la librería incluye *Hooks*[5], que facilitan y aclaran la sintaxis de las funcionalidades otorgadas por React.

Un punto de interés de React es el Document Object Model (DOM) virtual que mejora y flexibiliza el renderizado de la aplicación en el navegador, permitiendo obviar las partes no modificadas y evitar actualizaciones innecesarias.

El ecosistema de librerías adjuntas a React incluye múltiples librerías útiles para el desarrollo del proyecto, como puede ser **React Intl**, que permite la internacionalización de mensajes y fechas o **React Router** que simplifica los cambios de pantalla y actualiza la URL del navegador con cada cambio.

### 2.3.4 Redux

Redux es una librería de código abierto independiente a React que maneja el estado de la aplicación. Una de sus principales ventajas es su peso ligero y su facilidad de uso una vez superada la curva de aprendizaje.

Para utilizarlo en conjunto con React, en el proyecto se utiliza **react-redux**[6]. Utilizar la librería de esta manera permite extraer el estado de los componentes de React, manejándolo en sus propios componentes de forma modular, mejorando la estructura del código y la limpieza del mismo, haciendo la aplicación fácilmente extensible.

### 2.3.5 Moment.js

Esta inclusión se trata de una librería compatible con React que facilita la operación y obtención de fechas.

Aunque se encuentra en modo *legacy*[7, 8] y cuenta con múltiples alternativas mucho más optimizadas, esta librería se utiliza en la aplicación a la hora de operar con fechas. La librería fue seleccionada por su simpleza de utilización y su soporte en una gran cantidad de navegadores.

### 2.3.6 Bootstrap

Bootstrap es una biblioteca multiplataforma de código abierto mantenida por Twitter. Su uso en la aplicación es debido a que facilita el diseño de la interfaz mediante el uso de sus plantillas predefinidas y sistema de rejillas en filas y columnas.

Capítulo 3

# Metodología

Con el fin de completar los objetivos establecidos, deben seguirse unas pautas para un desarrollo correcto y sólido. En este capítulo se exploran en profundidad las elecciones de metodología del proyecto:

## 3.1 Introducción a metodologías ágiles

Se llama metodologías ágiles a aquellas que siguen o se adaptan a los valores y los doce principios del manifiesto ágil, redactado por un conjunto de 17 expertos en 2001[9].

Una de sus principales características es la sustitución del tradicional paradigma de desarrollo en cascada por una sucesión de iteraciones que mejoran y amplían el producto resultante. El desarrollo ágil da un peso inferior a la documentación en favor de comunicación interna por parte del equipo de trabajo.

Estas metodologías, ayudan a detectar errores y fallos de forma temprana en el desarrollo y promocionan una mejor adaptación al cambio. Mediante las iteraciones existe una mejor relación de *feedback* con el cliente y permite a los equipos auto-organizarse, mejorando su productividad y minimizando la carga de trabajo.

Los ejemplos más conocidos y utilizados de metodologías ágiles son Extreme Programming (XP), Scrum y Kanban.

## 3.2 Scrum

La metodología ágil **Scrum** está basada en iteraciones o *sprints* de duración limitada que encapsulan el diseño e implementación de funcionalidades requeridas para el proyecto, centrados en adaptaciones a los cambios que puedan surgir, desgranando los problemas a resolver en proyectos de tamaño más reducido. Scrum está enfocado a equipos, teniendo roles que se asignan a los miembros del equipo.

El primero de los roles es el *product owner*, que funciona como punto de contacto con el usuario final, su cometido es escribir las **historias de usuario**, asignarles prioridades y añadirlas al *product backlog*. El segundo rol de relevancia es el *scrum master*, encargado de arbitrar el cumplimiento de las reglas de la metodología. Por último, el rol de *developer* o desarrollador es cada miembro del equipo que se encarga de la implementación de un sprint y todas las fases asociadas al mismo.

La documentación en Scrum consiste principalmente del *product backlog* y el *sprint backlog*, siendo estos conjuntos de requisitos planteados como historias de usuario. Usualmente, estas historias de usuario utilizan la estructura *"Como <rol> quiero <funcionalidad> para <beneficio>"* y llevan asociado un valor llamado *puntos historia* que facilita la comparación de unas tareas con otras, además de simplificar y ayudar con una estimación inicial. El *product owner*, además de asignar esta valoración, puede añadir a la historia de usuario una o más condiciones de aceptación que cuando se cumplen, simbolizan la finalización de la historia de usuario.

Los sprints son incrementos de tiempo definidos por los miembros del equipo. Usualmente un periodo de dos semanas durante el cual se realizan dos tipos de reuniones principales, siendo estas la reunión de planificación del sprint y las reuniones diarias o *daily scrum meeting*. En la reunión de planificación del sprint, los desarrolladores toman del *product backlog* las descripciones realizadas en las historias de usuario que implementarán a lo largo del sprint, creando un *sprint backlog* conjunto. Por otra parte, las reuniones diarias son reuniones cortas que sirven como elemento de control, permitiendo a los desarrolladores plantear sus objetivos diarios.

## 3.3 Kanban

Kanban es una metodología de gestión de trabajo centrada en minimizar y controlar el trabajo en curso. Esta minimización se consigue mediante una ayuda visual, utilizando un tablero con tarjetas de funcionalidades divididas por su estado, ya sean funcionalidades pendientes, en curso o finalizadas.

Esta estrategia es valiosa a la hora de estimar el progreso realizado o pendiente y resulta sencillo de utilizar en conjunción con elementos de Scrum definidos en la sección anterior (sec. 3.2).

## 3.4 Metodología aplicada al proyecto

En el caso de este proyecto, se han aplicado principios provenientes tanto de Scrum como de Kanban para planificar y elaborar cada uno de los incrementos iterativos. Estos incrementos

han sido divididos en subfases, siendo estas fases análisis, diseño, implementación y pruebas.

Debido a que la realización del trabajo es individual, los elementos de trabajo en equipo de Scrum como la división en roles y las reuniones diarias o por sprint han sido ignoradas, pero se ha mantenido la división de funcionalidades en historias de usuario y su agrupamiento en un *product backlog*.

Aunque se ha ignorado la división de roles, algunos principios de la división mantuvieron su utilidad. Por ejemplo, al realizar el análisis, el planteamiento a la hora de definir y encontrar las funcionalidades deseadas es similar al que haría el dueño de un proyecto Scrum.

Del método Kanban se ha utilizado la representación visual por columnas, donde se han colocado las tareas pendientes de cada sprint en un tablero físico mediante tarjetas adhesivas, moviéndolas a las columnas de en progreso y finalizado cuando era necesario. Esta representación ha sido de gran ayuda a la hora de resolver y asegurarse de la completitud de cada iteración.

En el cierre de cada iteración, se hizo una revisión rápida de las historias de usuario finalizadas, comprobando que la implementación cumple las necesidades planteadas desde el punto de vista de un cliente interesado. Además, se seleccionaron las historias que conformarían la siguiente iteración, vaciando el tablero Kanban en su totalidad y posicionando las nuevas tareas como pendientes.

La división del desarrollo en iteraciones facilita la reutilización de código de casos similares pertenecientes a iteraciones anteriores, que en conjunción con las herramientas y tecnologías utilizadas (cap. 2) probó ser una decisión correcta a lo largo del desarrollo de la aplicación.

Capítulo 4

# Análisis

En este capítulo se describe la separación en módulos correspondientes a los requisitos funcionales, además de los actores, historias de usuario concretas cuyo concepto se expuso en el capítulo de metodología (cap. 3) y finalmente la división por iteraciones de dichas historias de usuario.

## 4.1 Requisitos funcionales

Teniendo en cuenta que el ámbito de actividad de una tienda de ocio incluye múltiples actividades relacionadas pero diferenciadas, para facilitar la tarea de definir las historias de usuario que resuelven el problema a abarcar, se hizo una definición inicial de requisitos o grupos de funcionalidad generalizados que incluyen las funcionalidades esperadas y mencionadas como algunos de los objetivos del proyecto (sec. 1.3).

- **Gestión de usuarios:** Incluye funcionalidades como registro, autenticación, actualización de perfiles, visualización de usuarios y registro controlado de nuevos encargados para tiendas concretas.
- **Gestión del catálogo:** Cuenta con toda funcionalidad relacionada con los productos, las instancias concretas de productos (de venta, venta de segunda mano o alquiler) y las categorías a las que estos van asociados, permitiendo operaciones Create, Read, Update and Destroy (CRUD).
- **Gestión del negocio:** Maneja las operaciones de compra y alquiler de productos, reservas de espacios y recuperación de los mismos, incluida también la cancelación de alquileres antes de que ocurran.
- **Gestión de localizaciones:** Incorpora las funcionalidades buscadas para gestionar las tiendas, sus eventos y sus espacios asociados, pudiendo realizar operaciones CRUD sobre los elementos mencionados.

Esta división será la distribución en carpetas y paquetes interna del proyecto, este tema se profundizará en el próximo capítulo de la memoria sobre la implementación.

## 4.2 Actores involucrados

Existen dos clasificaciones generales de usuario. El primer tipo son usuarios administradores, de los que se diferencian dos actores con capacidades de gestión de los elementos de la aplicación. El segundo tipo de usuario es el que corresponde con un perfil de cliente. A continuación se explica en más profundidad cada actor individual.

### 4.2.1 Administrador web

Actor gestor de la empresa, tiene acceso libre a la creación de cualquier tipo de evento, espacio, producto, instancia de producto o tienda, además de añadir de forma rápida empleados asociados a tiendas concretas.

Este actor puede comprobar también la información de pedidos y reservas realizadas por el actor cliente, pudiendo cancelar o marcar como pago cualquiera de las reservas del cliente.

### 4.2.2 Empleado de tienda

Actor gestor de un establecimiento concreto de la empresa, sigue siendo un perfil de administrador, pero está limitado y cuenta con las funcionalidades CRUD de productos, instancias de producto y eventos. Los dos últimos elementos mencionados, las instancias de producto y eventos creados, solo pueden estar asociados a su tienda adjunta, pudiendo controlar el almacén de su tienda individual, pero no alterar o añadir en aquellos establecimientos que no están bajo su control.

Este actor puede comprobar también la información de pedidos y reservas realizadas por el actor cliente, permitiendo así confirmar los pagos de los mismos o cancelar reservas de cualquier tipo.

En definitiva, este rol de empleado de tienda está limitado y no puede operar creando tiendas ni registrando otros empleados de tienda por su cuenta, mientras que el Administrador web dispone de todas las herramientas existentes para la aplicación sin limitaciones de ningún tipo.

### 4.2.3 Cliente

Usuarios promedio de la aplicación. Si deciden no identificarse, podrán buscar el catálogo de productos, eventos y comprobar información de las tiendas pero no podrán realizar reservas ni compras de ningún tipo. El momento en el que un usuario se registra e identifica, gana

acceso a la compra y alquiler de productos, la reserva de espacios y las herramientas de edición de su perfil. También obtienen acceso a la revisión de su historial de compras realizadas, productos alquilados y espacios reservados, pudiendo cancelar peticiones pendientes.

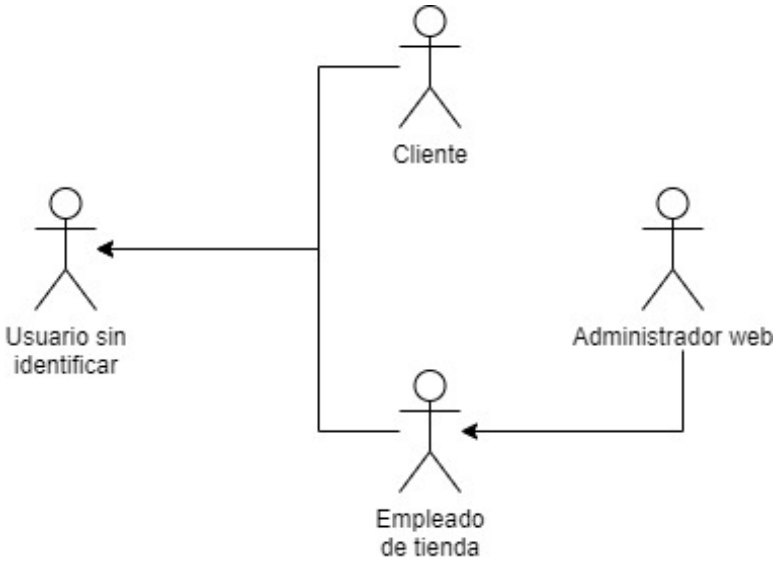


Figura 4.1: Diagrama de actores

## 4.3 Historias de usuario

A continuación se describen las historias de usuario asociadas a cada requisito funcional de la aplicación, siguiendo el formato *"Como <actor> quiero <funcionalidad> [para <beneficio>]"*. El apéndice sobre el beneficio es opcional en las siguientes descripciones ya que no siempre tiene sentido añadirlo.

### 4.3.1 Gestión de usuarios

#### Registro de cliente

Como usuario sin identificar quiero crear una cuenta para poder acceder a las funcionalidades que ofrece la aplicación y crear mi perfil de cliente.

#### Identificación de usuario

Como cliente, empleado de tienda y administrador web quiero poder identificarme para utilizar la aplicación.

#### Registro de empleados

Como administrador web quiero crear perfiles de empleado y asignarlos a una tienda para que los nuevos trabajadores de un establecimiento puedan trabajar y revisar encargos de alquiler, pedidos y otras tareas de gestión de sus establecimientos.

#### Ver perfil de usuario

Como administrador web o empleado de tienda quiero ver la información del perfil de otros usuarios registrados como su nombre de usuario, dirección de correo, etc.

#### Actualizar contraseña

Como usuario identificado de cualquier tipo quiero actualizar mi contraseña para mantener mi cuenta segura.

#### Ver mi perfil

Como usuario de cualquier tipo quiero ver la información de mi perfil para comprobar la información almacenada en la aplicación relativa a mi perfil de usuario.

#### Actualizar perfil

Como usuario identificado de cualquier tipo quiero actualizar mi perfil para corregir la información incorrecta del mismo.

#### Visualización de la lista de usuarios

Como administrador web o empleado de tienda quiero ver la lista de usuarios de la aplicación para buscar usuarios por su nombre en caso de que necesite ver su información.

### 4.3.2 Gestión de catálogo

#### Creación de categoría

Como administrador web o empleado de tienda quiero añadir categorías para agrupar los productos del catálogo y facilitar la búsqueda de los clientes interesados.

#### Modificación de categoría

Como administrador web o empleado de tienda quiero modificar la información de una categoría para corregir posibles errores.

**Eliminar categoría**

Como administrador web o empleado de tienda quiero poder eliminar categorías que ya no se utilizan para mantener la aplicación limpia.

**Añadir producto**

Como administrador web o empleado de tienda quiero añadir nuevos productos al catálogo para almacenar en la aplicación todos los productos disponibles en el almacén físico.

**Modificar producto**

Como administrador web o empleado de tienda quiero modificar productos del catálogo para asegurar que la información de los mismos es correcta y actualizada.

**Eliminar producto**

Como administrador web o empleado de tienda quiero eliminar productos que he introducido en la aplicación por error y que no hayan sido usados todavía para venta o alquiler con el fin de mantener la aplicación limpia.

**Búsqueda de productos**

Como usuario de cualquier tipo quiero buscar productos mediante parámetros opcionales como palabras clave o categoría para visualizar la sección del catálogo que cumpla con los parámetros opcionales establecidos para la búsqueda.

**Detalles de producto**

Como usuario de cualquier tipo quiero visualizar los detalles de un producto concreto para decidir si comprarlo en caso de ser un cliente o acceder a las opciones de gestión administrativas en caso de ser un administrador.

**Añadir stock de un producto**

Como administrador web o empleado de tienda quiero añadir una instancia o *stock* de un producto concreto en una tienda concreta para manejar el almacén.

**Modificar stock de un producto**

Como administrador web o empleado de tienda quiero modificar la información de una pieza de stock de un producto para cambiar el precio o el tipo de producto del que se trata.

#### Eliminar stock de un producto

Como administrador web o empleado de tienda quiero eliminar la instancia o *stock* de un producto para mantener la información del almacén físico igual a la información de la aplicación.

#### Ver stock disponible de un producto

Como cliente quiero ver el stock o copias disponibles para compra de un producto para añadirlas a mi carrito de la compra. Además, como administrador web o empleado de tienda quiero ver el stock disponible para comprobar la situación del almacén de un producto concreto.

#### Comprobar disponibilidad para alquiler de un producto

Como cliente quiero comprobar si el producto que busco alquilar se encuentra disponible en una tienda concreta durante el periodo de fechas estipulado.

#### Visualizar próximos lanzamientos

Como usuario generalizado de la aplicación quiero ver una lista de los cinco productos con su fecha de salida más cercana para estar al tanto de próximas actualizaciones al catálogo.

### 4.3.3 Gestión de localizaciones

#### Creación de tienda

Como administrador web quiero crear nuevas tiendas fácilmente para estar preparado para ampliar el tamaño de la empresa.

#### Modificación de tienda

Como administrador web o empleado de tienda asignado a la tienda quiero poder modificar una tienda bajo mi control para mantener la información correcta y actualizada.

#### Eliminación de tienda

Como administrador web quiero eliminar tiendas que he introducido por error y que no hayan sido utilizadas todavía para mantener la aplicación limpia y con información correcta.

**Visualizar tiendas**

Como usuario de cualquier tipo quiero visualizar las tiendas de una provincia concreta para ver su información y poder localizarlas rápidamente en caso de que necesite acudir a un establecimiento concreto.

**Ver detalle de tienda**

Como usuario de cualquier tipo quiero visualizar la información de una tienda concreta para ver su información y poder acudir en persona a recoger los productos alquilados, asistir a eventos u otros motivos de interés.

**Añadir espacios a una tienda**

Como administrador web o empleado de tienda quiero añadir nuevos espacios cuando sea necesario para aumentar el número de mesas disponibles en el establecimiento.

**Modificar espacios de una tienda**

Como administrador web o empleado de tienda quiero modificar espacios de una tienda cuando sea necesario para que la información sea correcta y se ajuste a la realidad del establecimiento.

**Eliminar espacios de una tienda**

Como administrador web o empleado de tienda quiero eliminar espacios de una tienda que he añadido por error y no cuenten con reservas de ningún tipo para mantener la aplicación con información correcta sobre los espacios.

**Comprobación de disponibilidad de espacios entre fechas**

Como usuario de cualquier tipo quiero comprobar la disponibilidad de los espacios de una tienda entre dos fechas concretas para ver cuales están ocupados o cuales puedo reservar para un rango de fechas estipulado.

**Visualización de eventos próximos de una tienda**

Como usuario de cualquier tipo quiero ver una lista de los eventos próximos o cercanos en una tienda concreta para estar al tanto de próximos eventos interesantes.

#### Búsqueda de eventos

Como usuario de cualquier tipo quiero buscar todos los eventos que cumplan alguno de los siguientes criterios opcionales: contenga en su título palabras clave, se realice en una tienda concreta u ocurra entre dos fechas seleccionadas. Los criterios deben ser combinables, pudiendo buscar eventos en tiendas concretas para fechas seleccionadas.

#### Creación de eventos

Como administrador web o empleado de tienda quiero añadir eventos a una tienda concreta para mantener a los usuarios informados de la planificación de eventos.

#### Actualización de eventos

Como administrador web o empleado de tienda quiero poder actualizar la información de un evento para ajustar alguno de sus campos a la realidad.

#### Eliminación de eventos

Como administrador web o empleado de tienda quiero poder eliminar eventos para evitar tener eventos innecesarios, repetidos o erróneos en la planificación de eventos de la aplicación.

### 4.3.4 Gestión de negocio

#### Añadir al carrito

Como cliente quiero añadir productos a mi carrito de la compra para poder comprar varios productos de manera simultánea y ver la información de mi próxima compra de forma cómoda.

#### Eliminar del carrito

Como cliente quiero eliminar productos de mi carrito de la compra cuando sea necesario para asegurarme de no comprar lo que no quiero comprar.

#### Confirmar compra del carrito

Como cliente quiero confirmar los productos de mi carrito para comprarlos seleccionando la tienda donde quiero recoger mi pedido y viendo un resultado final en forma de factura.

#### Confirmar pago de pedido de productos

Como administrador web o empleado de tienda quiero confirmar el pago de un pedido de compra de productos para que cuando el usuario venga a buscar su pedido o a pagarlo pueda marcarlo como pagado en el sistema.

#### Cancelar pedido de productos

Como administrador web o empleado de tienda quiero cancelar un pedido de compra de productos para que en el caso de que el usuario decida no recoger el pedido o surja cualquier inconveniente, pueda deshacerse la compra.

#### Ver información de pedido

Como usuario quiero ver los detalles de un pedido para el que tengo permisos para consultar la información de pago o recordar la tienda donde debo ir a recogerlo. En caso de un administrador puede que el beneficio que busco sea marcar el pedido como pagado o cancelarlo.

#### Visualización de pedidos

Como usuario quiero buscar mis pedidos de productos para ver un listado de mis compras pendientes y pasadas y como administrador quiero ver los pedidos de productos de cualquier usuario para comprobar su historial de compra.

#### Ver pedidos de producto pendientes

Como administrador web o administrador de tienda quiero ver los pedidos de producto que aún no han sido recogidos para marcarlos como pago o cancelado en caso de que sea necesario.

#### Alquilar un producto

Como cliente quiero alquilar un producto que se encuentra en una tienda entre unas fechas concretas para utilizarlo en el periodo designado.

#### Reservar un espacio

Como cliente quiero alquilar un espacio de una tienda para unas fechas concretas para asegurar mi sitio en un evento que ocurre en días concretos.

**Confirmar pago de alquiler de producto**

Como administrador web o empleado de tienda quiero confirmar el pago de un producto alquilado por un cliente para eliminarlo de la lista de pedidos pendientes.

**Cancelar un alquiler de producto**

Como administrador quiero cancelar un alquiler de un producto de un cliente para quitarlo de pedidos pendientes de pago en caso de que haya algún error en el proceso de alquiler. Como cliente quiero cancelar uno de mis propios alquileres para poder cambiar de opinión si ocurre cualquier imprevisto.

**Confirmar pago de reserva de espacio**

Como administrador web o empleado de tienda quiero confirmar el pago de un espacio alquilado por un cliente para quitarlo de la lista de espacios alquilados pendientes de pago.

**Cancelar reserva de espacio**

Como administrador quiero cancelar una reserva de espacio de un cliente para poder eliminarlo de la lista de alquileres pendientes si es necesario. Como cliente quiero cancelar una de mis propias reservas para poder cambiar de opinión sobre la reserva del espacio.

**Ver alquileres de producto pendientes**

Como administrador web o empleado de tienda quiero ver las reservas o alquileres de producto que aún no han sido pagados para marcarlos como pagados o cancelados en caso de que sea necesario.

**Ver reservas de producto**

Como administrador web o empleado de tienda quiero ver las reservas de productos de un usuario para revisar el historial de alquileres de producto cuando sea necesario. Como cliente quiero ver mis propias reservas de producto. Esto incluye ver los detalles de una reserva individual de producto.

**Ver reservas de espacio**

Como administrador web o empleado de tienda quiero ver las reservas de espacios de un usuario para revisar el historial de alquileres de espacio cuando sea necesario. Como cliente quiero ver mis propias reservas de espacio. Esto incluye ver los detalles de una reserva individual de espacio.

**Ver reservas de espacio pendientes**

Como administrador web o empleado de tienda quiero ver las reservas de espacios que aún no han sido pagadas para marcarlas como pagadas o canceladas en caso de que sea necesario.

## 4.4 Planificación de iteraciones

A continuación se expone la división en iteraciones planeada para la implementación del proyecto. En cada apartado se comentarán los objetivos de la iteración y cualquier otra explicación pertinente sobre la motivación de cada iteración. También se mencionan las historias de usuario que se implementan en cada iteración.

El objetivo de esta división es que cada iteración amplíe sobre la funcionalidad implementada por la iteración anterior, implementando primero los elementos críticos de los que dependen otras historias de usuario de la aplicación.

### 4.4.1 Iteración 1

La primera iteración tiene como objetivo empezar la base de datos y la estructura de ficheros básica del proyecto. Se busca principalmente establecer un esqueleto sobre el que trabajar y añadir las funcionalidades de próximas iteraciones.

Con el objetivo de simplificar el desarrollo, se partió de una base de código reutilizado compatible en algunas historias de usuario concretas. Este código sirvió como puente de contacto para la familiarización con el entorno de desarrollo y la distribución de ficheros de la aplicación.

La iteración 1 implementa las siguientes historias de usuario:

- **Gestión de catálogo** (sec. 4.3.2): Creación de categoría, modificación de categoría, eliminar categoría, añadir producto, modificar producto, eliminar producto, búsqueda de productos y detalles de producto.
- **Gestión de usuarios** (sec. 4.3.1): Registro de cliente, identificación de usuario, actualizar contraseña, actualizar perfil.

### 4.4.2 Iteración 2

La segunda iteración busca implementar el control del stock de los productos existentes, además de preparar una base para la futura compra de productos y mejorar la experiencia de usuario con una forma de visualizar y buscar próximos lanzamientos de productos.

Con esta iteración, la mayor parte de la gestión del catálogo queda finalizada, descontando modificaciones en futuras iteraciones para integrar nuevos cambios realizados.

La iteración 2 implementa las siguientes historias de usuario:

- **Gestión de catálogo** (sec. 4.3.2): Añadir stock de un producto, modificar stock de un producto, eliminar stock de un producto, ver stock disponible de un producto, visualizar próximos lanzamientos.

### 4.4.3 Iteración 3

La tercera iteración añade el control de los administradores sobre las tiendas y la visualización de las mismas para todo tipo de usuarios. En esta iteración, el usuario tendrá la posibilidad de consultar la lista de tiendas que forman parte de la empresa filtrando por provincias.

La iteración 3 implementa las siguientes historias de usuario:

- **Gestión de localizaciones** (sec. 4.3.3): Creación de tienda, modificación de tienda, eliminación de tienda, visualizar tiendas, ver detalle de tienda.

### 4.4.4 Iteración 4

La cuarta iteración tiene como objetivo implementar los métodos básicos de negocio relacionados con la compra de productos y el carrito de compra del cliente además de cualquier funcionalidad asociada al propio carrito. Las compras realizadas se agrupan en ventas que son mencionadas a lo largo del documento como *pedidos*.

La iteración 4 implementa las siguientes historias de usuario:

- **Gestión de negocio** (sec. 4.3.4): Añadir al carrito, eliminar del carrito, confirmar compra del carrito, visualización de pedidos, ver información de pedido, confirmar pago de pedido de productos, cancelar pedido de productos.

### 4.4.5 Iteración 5

La quinta iteración realiza las historias de usuario relacionadas con los alquileres de productos, pudiendo alquilar productos individuales según la tienda y las fechas que se hayan seleccionado.

La iteración 5 implementa las siguientes historias de usuario:

- **Gestión de catálogo** (sec. 4.3.2): Comprobar disponibilidad para alquiler de un producto.

- **Gestión de negocio** (sec. 4.3.4): Alquilar un producto, ver pedidos de producto pendientes, ver alquileres de producto pendientes, confirmar pago de alquiler de producto, cancelar un alquiler de producto, ver reservas de producto.

### 4.4.6 Iteración 6

La sexta iteración tiene por objetivo añadir los eventos a la aplicación, además de la gestión de los mismos por parte de los administradores y la búsqueda avanzada de eventos o la visualización de eventos semanales de una tienda concreta. En esta iteración se amplían los detalles de las tiendas para mostrar cómodamente los eventos semanales próximos.

La iteración 6 implementa las siguientes historias de usuario:

- **Gestión de localizaciones** (sec. 4.3.3): Creación de eventos, actualización de eventos, eliminación de eventos, visualización de eventos próximos de una tienda, búsqueda de eventos.

### 4.4.7 Iteración 7

La séptima iteración añade los espacios a las tiendas de la aplicación, manejando espacios individualizados para cada establecimiento. En esta iteración, el usuario puede buscar entre dos fechas concretas para ver la ocupación del establecimiento en el que está interesado, pudiendo comprobar o reservar lo que necesite.

Con esta iteración, las historias de usuario de la gestión de localizaciones queda finalizada.

La iteración 7 implementa las siguientes historias de usuario:

- **Gestión de localizaciones** (sec. 4.3.3): Añadir espacios a una tienda, modificar espacios de una tienda, eliminar espacios de una tienda, comprobación de disponibilidad de espacios entre fechas.
- **Gestión de negocio** (sec. 4.3.4): Reservar un espacio, ver reservas de espacio pendientes, ver reservas de espacio.

### 4.4.8 Iteración 8

La octava iteración completa las funcionalidades de pago y cancelación de reservas de espacio, además de añadir funcionalidades finales de usuario como el registro de empleados o la visualización de perfiles. En esta iteración se busca añadir toques finales y completar el resto de historias de usuario planificadas para el proyecto.

La iteración 8 implementa las siguientes historias de usuario:

- **Gestión de negocio** (sec. 4.3.4): Confirmar pago de reserva de espacio, cancelar reserva de espacio.
- **Gestión de usuarios** (sec. 4.3.1): Registro de empleados, visualización de la lista de usuarios, ver perfil de usuario.

Capítulo 5

# Diseño

En este capítulo se describe la arquitectura y las elecciones de diseño del proyecto a desarrollar.

## 5.1 Arquitectura

La aplicación se trata de una Single-Page Application (SPA) o *Aplicación de página única*. Este tipo de aplicaciones web tienen como objetivo conseguir transiciones rápidas entre sus ”páginas” o vistas, agilizando su uso y adaptándose a las peticiones del usuario de forma dinámica. Contrario a una aplicación web tradicional, donde cada página se carga en cada actualización, una SPA actualiza exclusivamente los componentes cuyo estado haya sido alterado.

Se ha decidido aplicar una arquitectura cliente-servidor, donde tanto el cliente como el servidor están divididos en múltiples capas. El servidor cuenta con capas que manejan las operaciones de lógica de negocio, el acceso a datos y un servicio web. Por otro lado, el cliente es una interfaz visual basada en componentes con una capa de acceso al backend, que muestra la información recuperada mediante peticiones al servidor (fig. 5.1).

Las principales ventajas de la arquitectura cliente-servidor para este proyecto son la escalabilidad, reusabilidad de código y encapsulación, conteniendo los errores a nivel de capa.

## 5.2 Servidor/Backend

El servidor está implementado en Java. Cuenta con una división por capas donde se diferencia entre la capa modelo y la capa de servicio web o REST.

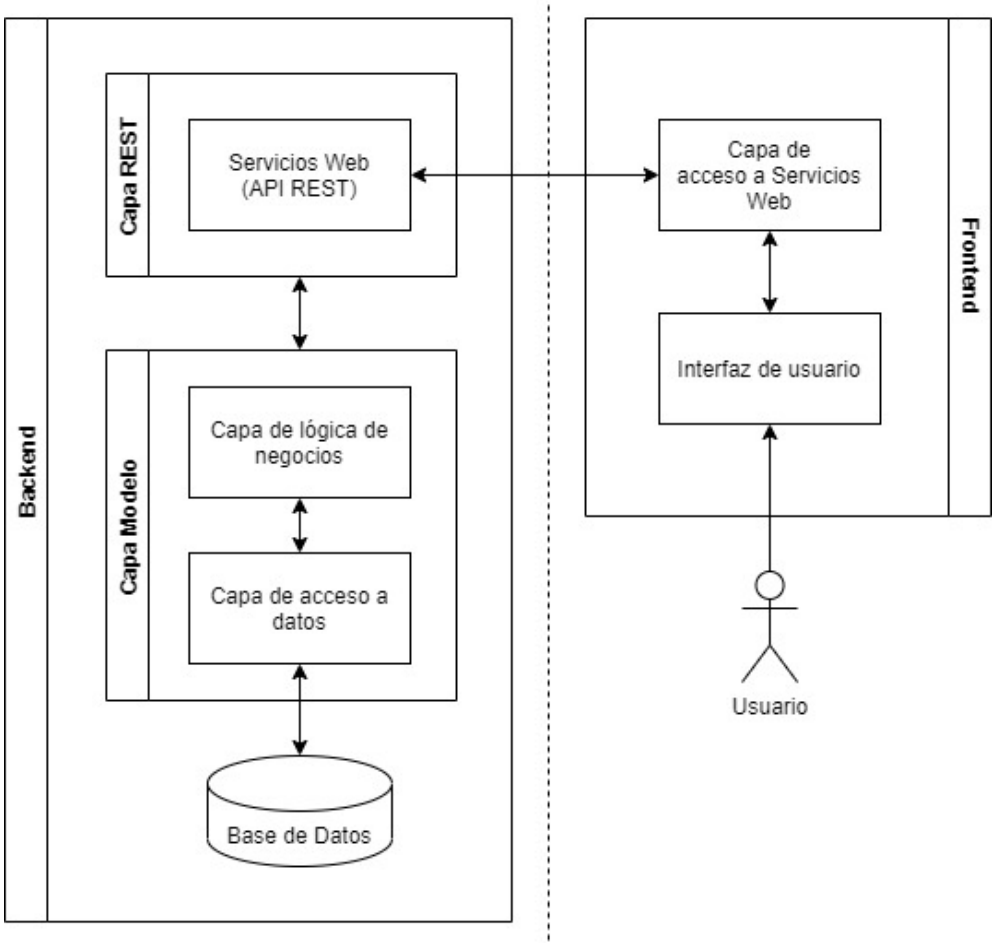


Figura 5.1: Arquitectura de la aplicación

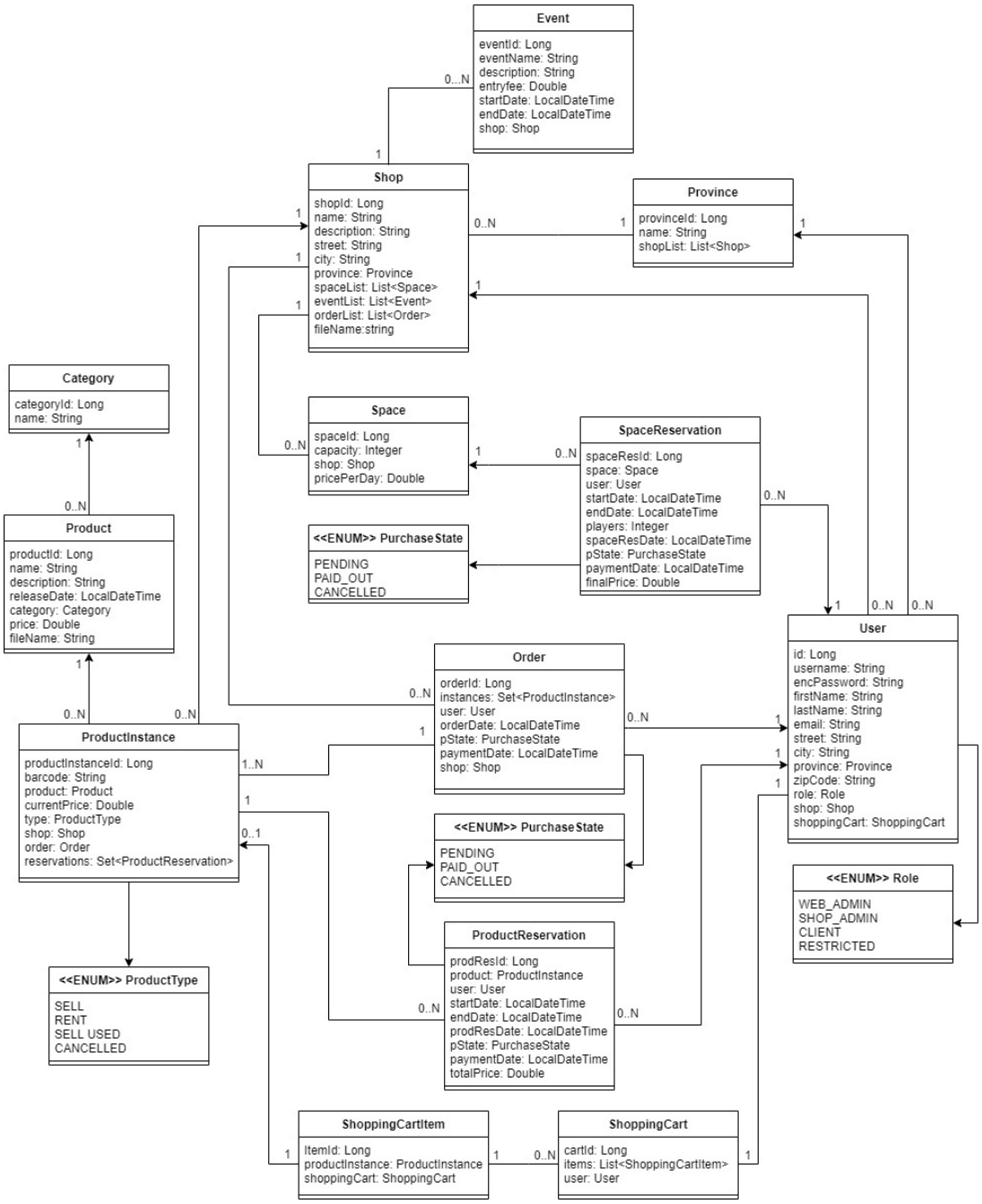


Figura 5.2: Entidades de la aplicación

### 5.2.1 Capa Modelo

La capa modelo está subdividida de nuevo en capas, siendo estas la capa de acceso a datos y la capa de lógica de negocio.

#### Capa de acceso a datos

La capa de acceso a datos contiene las clases que representan entidades de la base de datos y recupera la información de la misma siguiendo el patrón Data Access Object (DAO). En el contexto de la aplicación, esta capa implementa clases y DAOs para cada una de las entidades almacenadas en la base de datos. Una definición detallada de las relaciones entre las entidades de la base de datos que se utilizan se puede ver en la figura 5.2.

A continuación se muestran las entidades presentes en el modelo de la aplicación que deben hacerse persistentes en la base de datos, además se explica brevemente su propósito en el sistema.

- **Category:** Almacena la información de las categorías bajo las que se agrupan los productos del catálogo.
- **Event:** Almacena la información sobre los eventos ofrecidos por la empresa.
- **Order:** Almacena la información sobre los pedidos de productos realizados por los usuarios. Puede contener varios productos individuales. Cuenta con un campo para indicar su estado, pudiendo ser un pedido pendiente de pago, pagado o cancelado.
- **Product:** Almacena la información relevante sobre un producto del catálogo.
- **ProductInstance:** Almacena la información sobre las instancias existentes de un producto o stock. Cuenta con un tipo de producto que indica si es una instancia de venta, venta de segunda mano o alquiler. Con el fin de mantener un historial, existe un tipo de producto cancelado para manejar las cancelaciones de pedidos.
- **ProductReservation:** Almacena la información sobre el alquiler de una instancia de producto concreto para un rango de fechas definido. Cuenta con un campo para indicar su estado, pudiendo ser un alquiler pendiente de pago, pagado o cancelado.
- **Province:** Almacena la información sobre las provincias bajo las que se agrupan las tiendas.
- **Shop:** Almacena la información sobre cada tienda de la aplicación.
- **ShoppingCart:** Almacena la información sobre el carrito asociado a un usuario y su lista de ítems.

- **ShoppingCartItem:** Almacena la información sobre cada ítem individual de un carrito de la compra.
- **Space:** Almacena la información sobre un espacio perteneciente a una tienda.
- **SpaceReservation:** Almacena la información sobre las reservas de espacio de los usuarios para un rango de fechas concreto. Cuenta con un campo para indicar su estado, pudiendo ser una reserva pendiente de pago, pagada o cancelada.
- **User:** Almacena toda la información relevante para un usuario registrado. Cuenta con un campo para definir el rol del usuario, siendo este un cliente, administrador web o empleado de tienda.

Para cada una de las entidades mostradas se ha definido un DAO encargado de manejar la información presente en la base de datos siguiendo el patrón de diseño que se explica a continuación.

- **Patrón DAO:** El uso de este patrón aísla la base de datos concreta que se utiliza mediante los DAOs. Estos objetos manejan la información de cada entidad o tabla almacenada en la base de datos.

Para hacer el desarrollo de esta capa más sencillo, se utiliza Spring Data que genera automáticamente una implementación de los DAOs adaptándose al manejador de datos específico en el arranque de la aplicación, utilizando el nombre de los métodos declarados y los campos de la entidad.

Además, en la aplicación se implementan *PagingAndSortingRepository* de Spring Data para obtener acceso a las funcionalidades que otorga la librería como operaciones CRUD, operaciones de búsqueda básicas y paginación.

En ocasiones, como es el caso de algunas funcionalidades con parámetros opcionales, debemos implementar una búsqueda más específica de forma manual. Para solucionar esta situación se utilizan DAOs personalizados, como en la entidad *Product*, que se puede ver en la figura 5.3.

### Capa de lógica de negocio

La capa de lógica de negocio implementa servicios que contienen la funcionalidad buscada para la aplicación. Hace uso de la capa de acceso a datos a la hora de recuperar información necesaria para sus operaciones.

En la aplicación, la capa de lógica de negocio implementa un servicio para cada uno de los requisitos funcionales mencionados en el capítulo anterior (sec. 4.1). Cada servicio contiene

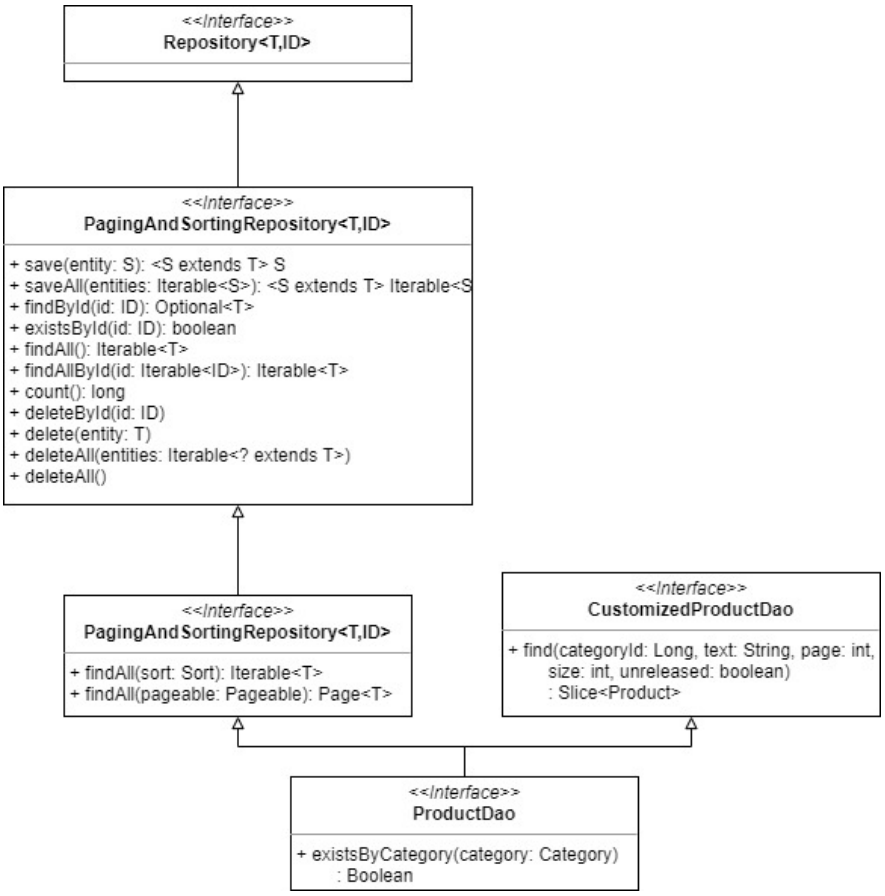


Figura 5.3: Ejemplo de DAO en la aplicación

métodos definidos con el objetivo de completar las historias de usuario concretas de cada requisito funcional generalizado.

El servicio que implementa la gestión de usuarios, *UserService* contiene toda la funcionalidad relevante para la gestión de usuarios de la aplicación. Desde el registro de usuarios a la búsqueda de un usuario, sus métodos se pueden ver en la figura 5.4.

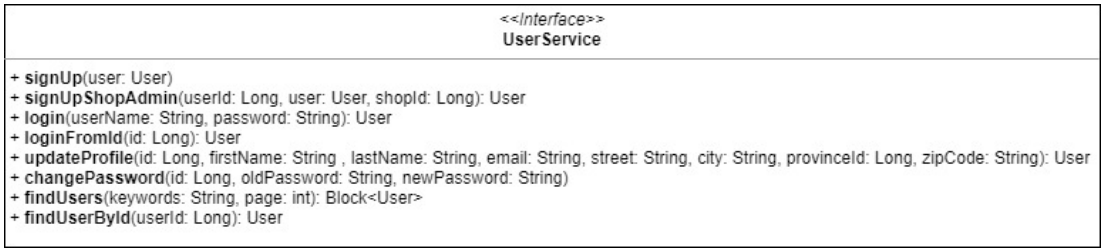


Figura 5.4: Métodos del servicio UserService

El servicio *CatalogService* implementa toda funcionalidad relacionada con la gestión del catálogo de productos de la aplicación. Los métodos implementados se encuentran en la figura 5.5.

El servicio de localizaciones o *LocationService* contiene métodos que operan con las localizaciones de la empresa como tiendas, espacios de las tiendas o eventos de las mismas. Un desglose de sus métodos se puede ver en la figura 5.6.

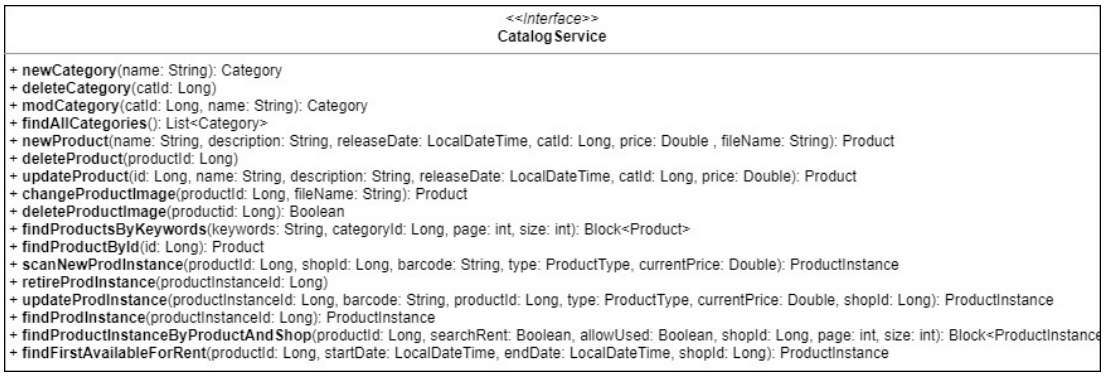


Figura 5.5: Métodos del servicio CatalogService

El último servicio expuesto se llama *RetailService*. Este servicio implementa métodos relacionados con las compras de productos y alquileres de espacios o productos como se pueden ver en la figura 5.7.

Existe un servicio que no se expone al exterior, encargado de las comprobaciones de permisos de usuario para los métodos restringidos según los actores definidos en la sección 4.3.1. Este servicio se llama *PermissionChecker* y se puede ver en la figura 5.8.

Las operaciones de los servicios pueden fallar en su ejecución, con la intención de contro-

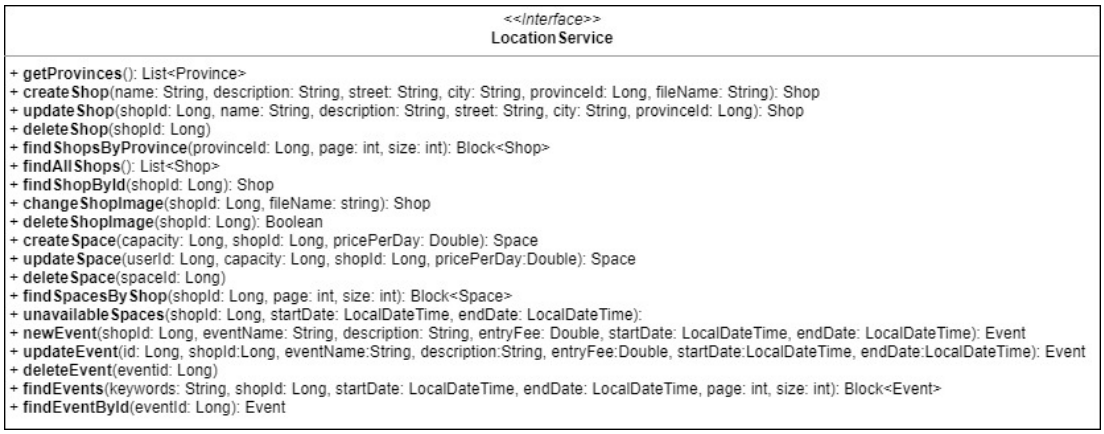


Figura 5.6: Métodos del servicio LocationService

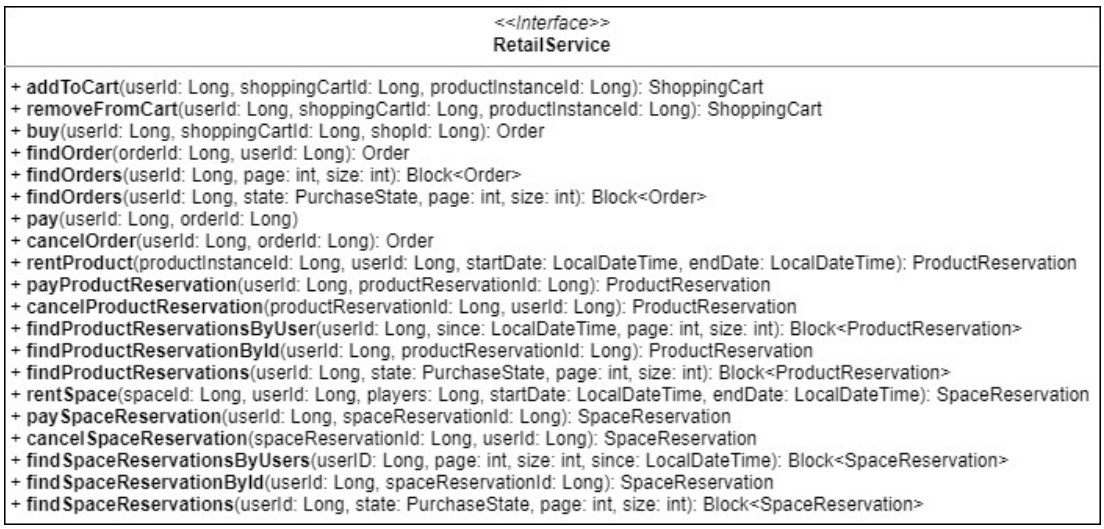


Figura 5.7: Métodos del servicio RetailService

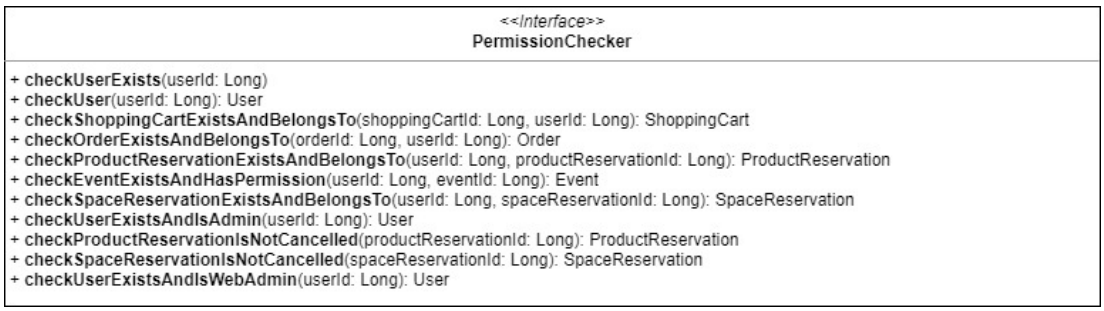


Figura 5.8: Métodos del servicio PermissionChecker

lar estos errores se utilizan excepciones con nombres descriptivos sobre el error. Por brevedad, a continuación se mostrarán y explicarán algunos ejemplos de entre las excepciones de la aplicación.

- **CancelledRentalException:** Esta excepción ocurre cuando se intenta pagar o cancelar un alquiler de un producto que ya ha sido cancelado con anterioridad.
- **CartFullException:** Esta excepción ocurre cuando se intenta añadir un producto al carrito de la compra y se alcanza la capacidad máxima establecida.
- **DuplicateInstanceException:** Esta excepción ocurre cuando se intenta añadir una entidad de la que ya existe una instancia con la misma clave primaria.
- **IncorrectDateException:** Esta excepción ocurre cuando se intenta insertar un rango de fechas incorrecto. Por ejemplo, cuando una fecha final ocurre con anterioridad a la fecha inicial.
- **InstanceNotFoundException:** Esta excepción ocurre cuando se intenta recuperar una entidad concreta y no está presente en los datos de la aplicación.
- **NotForSaleException:** Esta excepción ocurre cuando una operación intenta comprar una instancia de producto destinada a alquiler.
- **PermissionException:** Esta excepción ocurre cuando un usuario sin permisos intenta acceder a una operación para la que no debería tener acceso dependiendo de su rol de usuario.
- **ProductUnavailableException:** Esta excepción ocurre cuando se intenta reservar un producto para alquiler y no se encuentra disponible para las fechas indicadas.

A continuación se explican otros aspectos sobre la implementación de la capa de lógica de negocio como el patrón fachada o el paradigma de inyección de dependencias.

- **Patrón fachada:** Este patrón de diseño nos permite ocultar la implementación propia de la aplicación, exponiendo una interfaz con métodos que representan funcionalidades concretas. Un ejemplo del patrón se puede ver en la figura 5.9.

  Esta fachada permite ocultar las particularidades de implementación de la lógica de negocio y las entidades que maneja la capa de acceso a datos, encargándose de completar las operaciones requeridas por las interacciones del usuario.
- **Inyección de dependencias:** Este es un patrón de diseño orientado a objetos que se utiliza para permitir a los servicios acceso a los DAOs (u otros servicios) que necesiten para realizar sus operaciones.

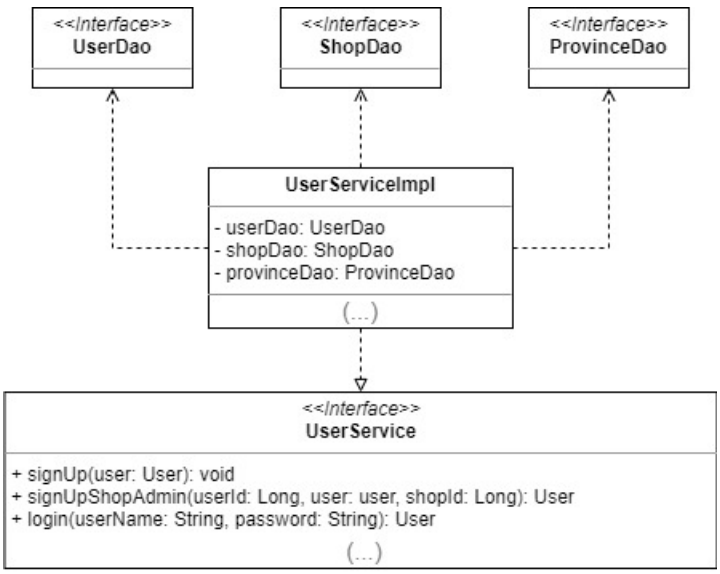


Figura 5.9: Ejemplo del patrón fachada

En el momento de arranque del servidor, Spring inicializa una instancia de los DAOs y de los servicios anotados con *@Service*. A esta instancia creada se le llama *bean* y se inyecta en cada uno de los atributos de su mismo tipo que estén marcados con la anotación *@Autowired*.

### 5.2.2 Capa REST

La capa de servicio web maneja las peticiones REST que se reciben desde el frontend. Una vez recibe la petición, delega la funcionalidad en la capa modelo y transforma los datos recibidos como objetos en Data Transfer Object (DTO).

#### Patrón DTO

En ocasiones, la información resultante de las operaciones en la capa modelo puede contener información que no queremos que sea vista por el usuario o información innecesaria, por tanto se implementan DTOs. Estos DTOs se encargan de reunir campos de una o varias entidades, pudiendo crear respuestas adaptadas que se envían o se reciben del cliente de la aplicación.

Contener la información en un objeto nos permite reducir el número de llamadas entre el cliente y el servidor, almacenando los datos que tradicionalmente se intercambiarían en peticiones individuales como una petición única.

#### Servicios REST

La comunicación con el frontend se realiza mediante peticiones REST, estas peticiones las recibe un servicio o controlador REST. La aplicación cuenta con una serie de controladores implementados utilizando la librería de Spring llamada Spring Web.

Cada uno de estos controladores otorga acceso al usuario a las operaciones públicas previamente expuestas en la capa de lógica de negocio. De esta forma existen cuatro controladores en la aplicación, *UserController*, *CatalogController*, *LocationController* y *RetailController*. A modo de ejemplo, se pueden ver las operaciones que realiza CatalogController en la figura 5.10.

Cada vez que se recibe una petición REST, se compara la ruta de la petición con la ruta única que identifica cada operación individualizada del controlador. Cada una de estas operaciones expuestas está anotada con un método de petición HTTP que indica el tipo de acción a realizar, correspondiéndose con los métodos de HTTP GET, POST, PUT y DELETE.

Una vez realizadas las operaciones correspondientes a la petición recibida, el controlador se encarga de realizar una respuesta en la que se adjunta la información necesaria para el usuario. Esta respuesta contiene un código de respuesta HTTP que resume el resultado de la operación, devolviendo códigos correspondientes a error en caso de un error o códigos de éxito en caso de que la operación se haya realizado correctamente.

Estos controladores utilizan el patrón de inyección de dependencias explicado con anterioridad en la sección 5.2.1, anotando cada uno de los controladores con *@RestController* e inyectando el servicio del que depende mediante *@Autowired*.

## 5.3 Cliente/Frontend

El cliente de la aplicación está implementado en Javascript mediante el uso de React y Redux. Está dividido en dos capas siendo estas la capa de acceso a servicio REST y la interfaz de usuario.

### 5.3.1 Capa de acceso a servicio REST

Esta capa incluye las llamadas a la capa REST del lado servidor que son necesarias para cubrir las acciones del usuario en el lado cliente. Estas llamadas permiten recuperar la información necesaria mediante el patrón Data Transfer Object (DTO) que el servidor le proporciona como respuesta.

En la aplicación se implementa un fichero Javascript para cada controlador REST del lado servidor (sec. 5.2.2) que define una o varias funciones para cada método expuesto por el controlador. De esta forma, para cada uno de los cuatro controladores accesibles por el usuario

| Método | Dirección recurso | Entrada | Salida |
|---|---|---|---|
| POST | /catalog/categories | categoryParams [body] | CategoryDto |
| DELETE | /catalog/categories/{categoryId} | categoryId [path] | void |
| PUT | /catalog/categories/{categoryId} | categoryId [path],<br>categoryParams [body] | CategoryDto |
| GET | /catalog/categories | - | List<CategoryDto> |
| POST | /catalog/products | userId [jwt], name [param],<br>description [param],<br>releaseDate [param],<br>categoryId [param],<br>price [param], file [param] | ProductDto |
| DELETE | /catalog/products/{productId} | userId [jwt], productId [path] | void |
| PUT | /catalog/products/productId | userId [jwt], productId [path]<br>productParams [body] | ProductDto |
| GET | /catalog/products/{productId} | productId [path] | ProductDto |
| GET | /catalog/products | categoryId [param],<br>keywords [param], page [param],<br>size [param], unreleased [param] | BlockDto<ProductDto> |
| PUT | /catalog/products/{productId}/changeImage | userId [jwt], productId [path]<br>file [param] | ProductDto |
| POST | /catalog/products/{productId}/productInstances | userId [jwt], productId [path],<br>productInstanceParams [body] | ProductInstanceDto |
| DELETE | /catalog/products/{productId}/productInstances<br>/{productInstanceId} | userId [jwt], productId [path]<br>productInstanceId [path] | void |
| PUT | /catalog/products/{productId}/productInstances<br>/{productInstanceId} | userId [jwt], productId [path],<br>productInstanceId [path],<br>productInstanceParams [body] | ProductInstanceDto |
| GET | /catalog/productInstances/{productInstanceId} | productInstanceId [path] | ProductInstanceDto |
| GET | /catalog/products/{productId}/instances | productId [path],<br>page [param] | BlockDto<ProductInstanceDto> |
| GET | /catalog/products/{productId}/instances | productId [path],<br>searchRent [param],<br>allowUsed[param], page [param] | BlockDto<ProductInstanceDto> |
| GET | /catalog/products/{productId}/instances | productId [path],<br>startDate [param],<br>endDate [param], shopId [param] | ProductInstanceDto |

Figura 5.10: Operaciones REST de CatalogController

tenemos un fichero de acceso a servicio homónimo. Un ejemplo de este tipo de fichero Javascript se puede ver en la figura 5.11.

Específicamente, *userService.js*, se corresponde con el controlador *UserController*. El fichero *catalogService.js*, se corresponde con el controlador *CatalogController*. El fichero *locationService.js*, se corresponde con el controlador *LocationController* y por último, el fichero *retailService.js*, se corresponde con *retailController*.

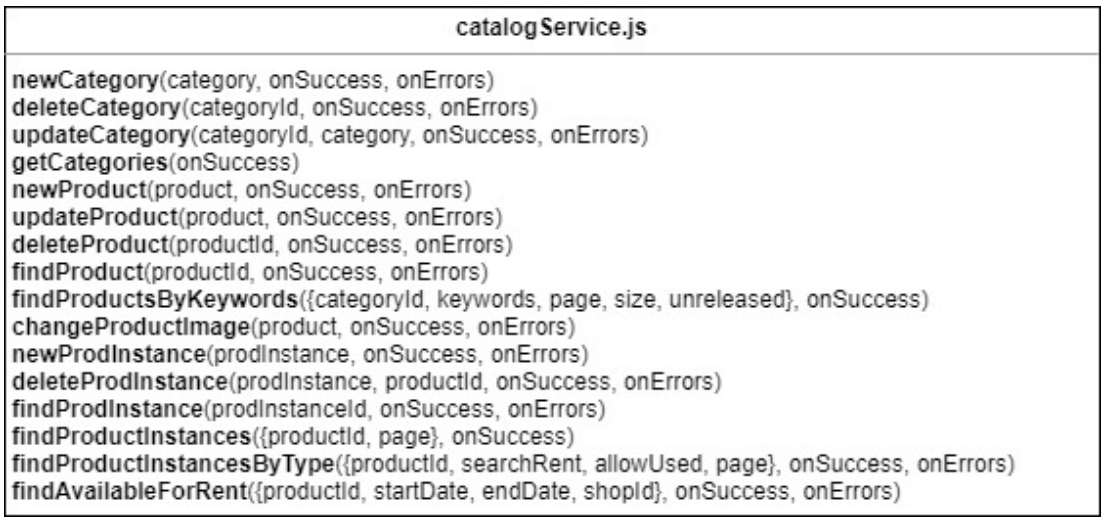


Figura 5.11: Funciones de catalogService.js

### 5.3.2 Interfaz de usuario

La última capa de la aplicación es el punto de contacto con el usuario final. La interfaz de usuario toma la información recuperada mediante la API REST y la muestra de forma comprensible para el usuario además de permitirle acceso a otras funcionalidades de la aplicación.

A continuación veremos un par de cuestiones de interés con respecto al diseño de la interfaz de usuario.

#### Organización en módulos

Para facilitar el mantenimiento y legibilidad del código, este se ha organizado en módulos que comparten funcionalidad y contienen cada uno de los componentes relacionados a dicha funcionalidad.

En el caso de esta aplicación, cada módulo sigue la distribución que se ha visto hasta ahora en otras secciones del proyecto, siendo estos módulos *user*, *catalog*, *location* y *retail*. Además, existen dos módulos que no se corresponden con ningún componente directo del lado servidor, estos son *app* y *common*.

- **User:** Este módulo contiene todos los elementos necesarios para la gestión del estado y muestra de información relacionadas con la gestión de los usuarios.
- **Catalog:** Este módulo contiene todos los elementos necesarios para la gestión del estado y muestra de información relacionadas con la gestión de los productos y almacén de los mismos.
- **Location:** Este módulo contiene todos los elementos necesarios para la gestión del estado y muestra de información relacionadas con la gestión de las tiendas, espacios y eventos.
- **Retail:** Este módulo contiene todos los elementos necesarios para la gestión del estado y muestra de información relacionadas con la gestión de las compras de productos y alquileres de todo tipo.
- **App:** Este módulo contiene cada uno de los elementos correspondientes a la estructura general de la interfaz como puede ser el encabezado o *header*, el *body* o el *footer* de la aplicación.
- **Common:** Este módulo incluye componentes reutilizables por el resto de módulos de la aplicación.

#### Mockups y diseño por componentes

En algunos casos complejos se han realizado mockups. Estas maquetas sirven para ver una simulación cercana al resultado final de las pantallas complejas y que componentes serán necesarios para obtener un resultado similar a ellas.

En la figura 5.12 se distinguen a la izquierda de la imagen los componentes estructurales principales de la aplicación, estos elementos se definen en el módulo *app*. Hacia arriba y abajo se indican los nombres de los componentes y sus respectivos componentes internos que forman parte de la pantalla de inicio de la aplicación.

En la figura 5.13 se muestra la vista de los detalles de una tienda, incluyendo tanto componentes para la visualización de eventos semanales, como un componente para comprobar la disponibilidad de los espacios propios de la tienda. Esta pantalla muestra funciones de edición, propias de un administrador web o empleado de la tienda.

#### Diseño web adaptable

En la aplicación, la interfaz de usuario ha sido diseñada con la intención de ser accesible desde cualquier tipo de dispositivo. Teniendo en cuenta el ascenso en popularidad de la navegación móvil, la aplicación tiene la intención de adaptarse a esta tendencia implementando una interfaz con formato *responsive*.

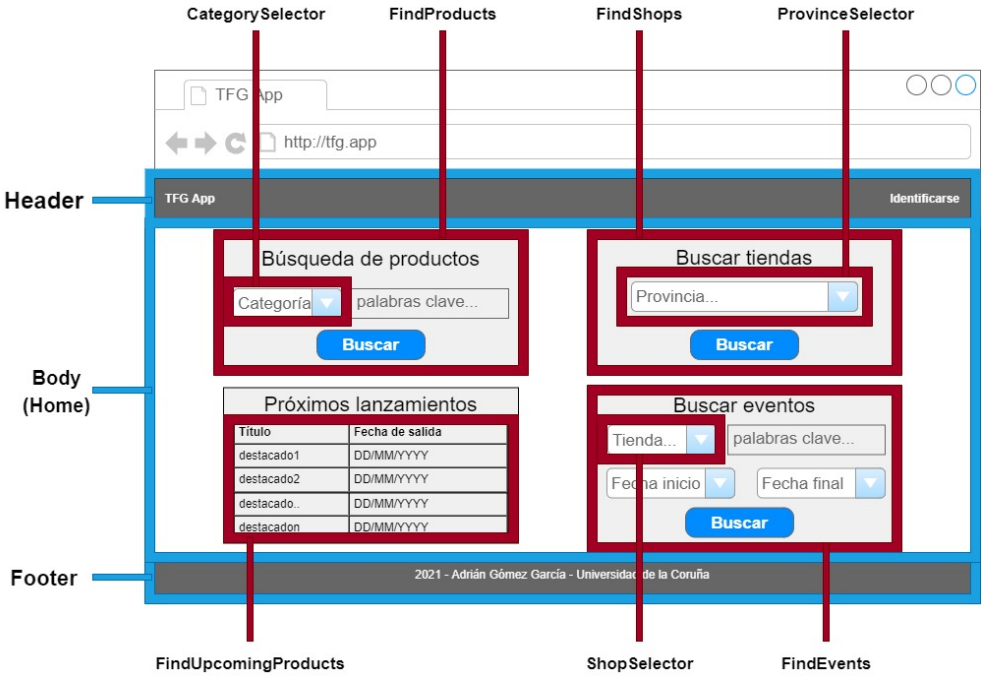


Figura 5.12: Mockup de la pantalla de inicio

Para lograr esta adaptación, en el proyecto se utiliza el framework *Bootstrap* que facilita la colocación por filas y columnas de los componentes de la aplicación y ofrece un diseño adaptable independientemente de la resolución del dispositivo como se ve en la figura 5.14.

### Internacionalización

La internacionalización consiste en otorgar soporte a múltiples lenguajes sin necesidad de alterar el código. Para conseguir este objetivo, los mensajes mostrados al usuario se encapsulan en ficheros de internacionalización a los cuales se acude dependiendo del idioma predeterminado en el navegador del usuario.

En la aplicación, esto se realiza mediante el uso del *bean* de Spring *MessageSource* en el backend y la librería de internacionalización *React Intl* para los mensajes propios del frontend. En el diseño actual no se traducen de ninguna forma las entradas de texto otorgadas por los usuarios (como puede ser el nombre de un producto, la descripción de una tienda, etc) dejándolo como una futura línea de trabajo para la mejora de la aplicación.

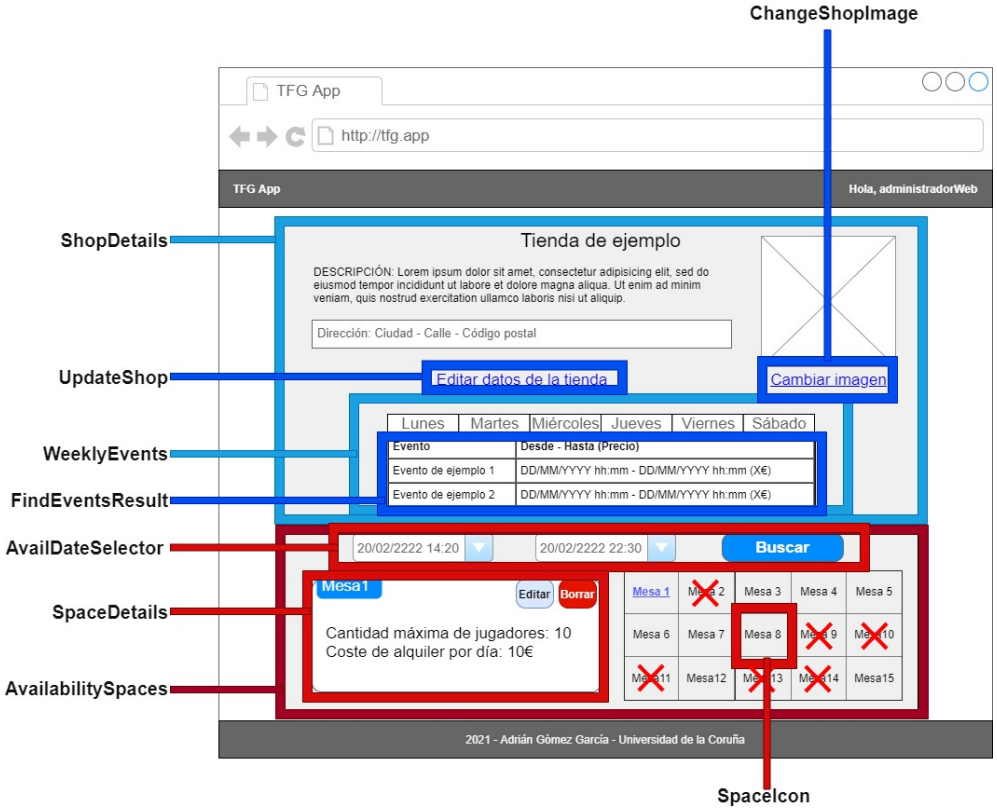


Figura 5.13: Mockup de los detalles completos de una tienda

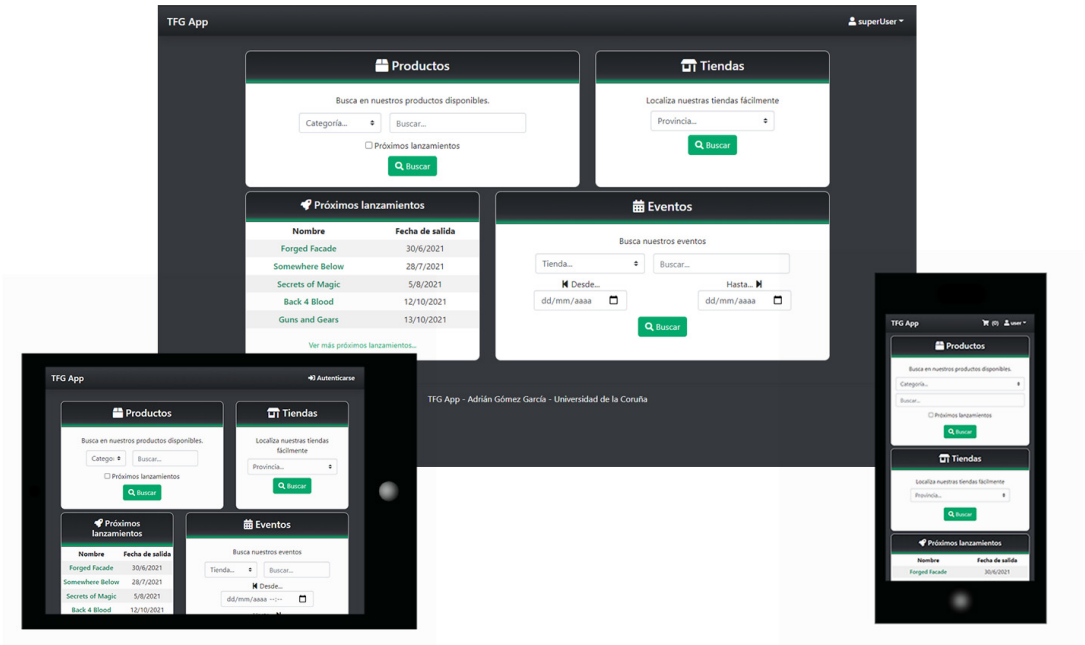


Figura 5.14: Ejemplos de distintas resoluciones

Capítulo 6

# Implementación

Este capítulo mostrará una vista detallada del proceso de implementación además de otros decisiones tomadas en las iteraciones realizadas. Por sencillez y organización de la memoria, en esta sección se explicarán las cuestiones relevantes de implementación empezando desde los niveles más bajos de la aplicación contenidos en el lado servidor y avanzando a los más altos del lado cliente.

## 6.1 Implementación del backend

En esta sección de la memoria se tratan cuestiones de implementación de cada una de las capas del backend, siendo estas la capa de acceso a datos, la capa de lógica de negocio y la capa REST.

### 6.1.1 Capa de acceso a datos

#### Implementación de las entidades

Para la implementación de las entidades se utiliza JPA/Hibernate. Este mapeador se encarga de identificar y asociar cada una de las entidades presentes en la base de datos a los objetos de entidad definidos en la capa de acceso a datos. Hibernate proporciona múltiples anotaciones para marcar las entidades de dominio.

Las entidades son identificadas por Hibernate mediante la anotación *@Entity*. Cualquier objeto anotado con *@Entity* se mapea por defecto con una tabla de la base de datos que comparta su nombre, siguiendo el mismo comportamiento para cada columna individual de la base de datos con los atributos del objeto anotado.

Existen casos donde el nombre del objeto no coincide exactamente con el de la entidad en la base de datos, para esas situaciones se utiliza la anotación *@Table*. Puede ocurrir también que el nombre de una columna de la base de datos no se corresponda con el nombre del

atributo en el objeto de entidad, en ese caso se utiliza la anotación *@Column*

Se utiliza la anotación *@Id* en conjunción con *@GenerateValue* para identificar la clave primaria del objeto y que se genere de forma automática siguiendo el criterio de generación establecido. En el caso concreto de la aplicación se generan identificadores de forma automática siguiendo la estrategia de generación IDENTITY definida por JPA/Hibernate.

Para las relaciones entre las entidades se utilizan varias anotaciones, siendo estas *@ManyToOne*, *@OneToMany* y *@OneToOne*. En el caso de las relaciones 1:N (fig. 5.2), la anotación *@ManyToOne* se utiliza para navegar desde el lado de la relación con cardinalidad N al lado de cardinalidad 1, en el caso inverso se utiliza la anotación *@OneToMany*. Para modelar correctamente la relación entre un usuario y su carro de la compra asociado se utiliza la anotación *@OneToOne* tanto en el usuario como en el carrito.

Tanto en las relaciones anotadas con *@OneToOne* como las relaciones anotadas como *@ManyToOne* se utiliza también la anotación *@JoinColumn* para indicar el nombre de la columna que representa la clave foránea.

### Implementación de los DAOs

Para facilitar la implementación de los DAOs de la aplicación se utiliza Spring Data que se encarga de implementar automáticamente la mayoría de los métodos en el momento de ejecución de la aplicación. Todos los DAOs definidos en la aplicación implementan *PagingAndSortingRepository* para contar con los métodos CRUD que ofrece, además de paginación.

Para muchas de las operaciones de búsqueda, se siguen convenciones de nombrado de Spring Data. Para ilustrar de una forma visual el concepto, se muestra a continuación la implementación en código de uno de los DAOs de la aplicación que usa estas convenciones de nombrado.

```
1 public interface OrderDao extends PagingAndSortingRepository<Order,
      Long> {
2   Slice<Order> findByUserIdOrderByOrderDateDesc(Long userId,
      Pageable pageable);
3   Slice<Order>
      findByPurchaseStateOrderByOrderDateDesc(PurchaseState
      purchaseState, Pageable pageable);
4 }
```

El código mostrado es la definición mediante convenciones de nombrado de Spring Data de dos operaciones de búsqueda necesarias para recuperar la información de las compras de productos de un usuario concreto y otra búsqueda por estado de compra (pendiente de pago, pagado, cancelado...). En este ejemplo se puede ver un atributo de tipo *Pageable*, este atributo recibe un objeto que indica la página concreta y el tamaño en elementos de la paginación que se debe aplicar a la información recuperada.

En ocasiones, las peticiones resultan más complejas o cuentan con particularidades como palabras clave que deben ser comprobadas individualmente o parámetros opcionales y no se pueden cubrir con estas convenciones de nombrado. Para solucionar este problema se implementan DAOs personalizados.

Estos DAOs personalizados son interfaces cuyos métodos deben ser implementados. En el caso de la aplicación estos ficheros siguen una nomenclatura con la forma *CustomizedEntityDao* para las interfaces y *CustomizedEntityDaoImpl* para las clases que las implementan.

Los DAOs personalizados incluyen métodos definidos de forma programática utilizando JPQL. A continuación se muestra el código de uno de los métodos definidos en la aplicación, este se encuentra incluido en *CustomizedProductDao* y permite la búsqueda paginada de productos y su filtración por palabras clave además de poder filtrar opcionalmente por categoría o solo mostrar próximos lanzamientos. Si no se especifica una categoría, el método debe devolver un resultado paginado de productos de todas las categorías disponibles.

```
1  public Slice<Product> find(Long categoryId, String text, int
     page, int size, boolean unreleased) {
2
3    String[] keywords = text == null ? new String[0] :
     text.split("\\s");
4    String queryString = "SELECT p FROM Product p WHERE ";
5
6    if (!unreleased)
7      queryString += "p.releaseDate < :comparenow";
8    else
9      queryString += "p.releaseDate >= :comparenow";
10
11   if (categoryId != null) {
12     queryString += " AND ";
13     queryString += "p.category.id = :categoryId";
14   }
15
16   if (keywords.length != 0) {
17     queryString += " AND ";
18
19     for (int i = 0; i < keywords.length - 1; i++) {
20       queryString += "LOWER(p.name) LIKE LOWER(:keyword" + i + ")
     AND ";
21     }
22
23     queryString += "LOWER(p.name) LIKE LOWER(:keyword" +
     (keywords.length - 1) + ")";
24   }
25
26   if (!unreleased)
```

```
27        queryString += " ORDER BY p.name";
28      else
29        queryString += " ORDER BY p.releaseDate";
30
31      Query query =
        entityManager.createQuery(queryString).setFirstResult(page *
        size).setMaxResults(size + 1);
32
33      LocalDateTime date = LocalDateTime.now();
34      query.setParameter("comparenow", date);
35
36      if (categoryId != null) {
37        query.setParameter("categoryId", categoryId);
38      }
39
40      if (keywords.length != 0) {
41        for (int i = 0; i < keywords.length; i++) {
42          query.setParameter("keyword" + i, '%' + keywords[i] + '%');
43        }
44
45      }
46
47      List<Product> products = query.getResultList();
48                                  (...)
49    }
```

### 6.1.2 Capa de lógica de negocio

#### Inyección de dependencias

Como se mencionó con anterioridad en el capítulo de diseño (pág. 37), en la aplicación se aplica el paradigma de inyección de dependencias mediante anotaciones de Spring.

Cada uno de los servicios de la capa de lógica de negocio necesita tener acceso a ciertos DAOs, para ello se inyecta un bean que implementa la interfaz del propio DAO utilizando la anotación *@Autowired*.

A su vez, cada servicio tiene que estar accesible para los controladores de la capa REST, para ello se anotan con *@Service*.

A continuación se muestra un fragmento de código de la implementación del servicio LocationServiceImpl que incluye estas anotaciones.

```
1     @Service
2     @Transactional
3     public class LocationServiceImpl implements LocationService {
4
```

```
5       @Autowired
6       private ProvinceDao provinceDao;
7
8       @Autowired
9       private ShopDao shopDao;
10                                  (...)
11    }
```

### Transaccionalidad

Las anotaciones de Spring también permiten gestionar de forma declarativa la implementación de la transaccionalidad. Esto se especifica mediante la anotación *@Transactional*. Un ejemplo de la anotación se puede ver en la subsección anterior de la memoria (sec. 6.1.2).

Esta anotación se puede utilizar a nivel de clase o a nivel de método concreto. Cuando se utiliza la anotación, permite que en el momento de llamada a uno de los métodos del modelo, si la llamada se produce en el contexto de una transacción ya en curso, se enganche a ella. Por otra parte, si la llamada ocurre fuera de un contexto de transacción, se creará una nueva transacción.

Durante la ejecución del método llamado, toda operación invocada desde los DAOs u otros servicios se enganchan a esta transacción. En un caso exitoso, la transacción termina con un *commit*. Si se lanza una excepción durante la transacción, esta transacción se finaliza con un *rollback*.

En casos concretos de métodos de solo lectura se utiliza *@Transactional(readOnly=true)* permitiendo que el gestor de transacciones realice optimizaciones.

En el contexto de la aplicación, *@Transactional* se utiliza a nivel de clase para los cuatro servicios implementados en la capa de lógica de negocio. En los casos de métodos concretos que solamente recuperan información con la intención de mostrarla, se ha utilizado la anotación *@Transactional(readOnly = true)*. Como ejemplo, a continuación se puede ver el código correspondiente al método *findOrder* del servicio *RetailService*.

```
1  @Transactional(readOnly = true)
2  public Order findOrder(Long orderId, Long userId) throws
     InstanceNotFoundException, PermissionException {
3
4    return permissionChecker.checkOrderExistsAndBelongsTo(orderId,
     userId);
5  }
```

### 6.1.3 Capa REST

Para manejar las peticiones REST recibidas del frontend, se implementan cuatro controladores, uno por cada servicio de la capa de lógica de negocio. Para facilitar la implementación de los mismos se utiliza la librería Spring Web.

#### Inyección de dependencias

En cada uno de los controladores implementados para manejar las peticiones REST necesita utilizar las funciones expuestas por los servicios de la capa de lógica de negocio. Para acceder a estas funciones se usa de nuevo el paradigma de inyección de dependencias, empleando la anotación *@RestController*. Además, se anotan los servicios importados desde la capa de lógica de negocio con *@Autowired*.

```
1  @RestController
2  @RequestMapping("/shop")
3  public class RetailController {
4
5    @Autowired
6    private RetailService retailService;
7
8    @Autowired
9    private MessageSource messageSource;
10
11                                (...)
12 }
```

#### Mapeado de peticiones HTTP REST

Para especificar el mapeado de las peticiones recibidas se utilizan varias anotaciones. Primero, *@RequestMapping* se utiliza para especificar la dirección correspondiente al controlador que precederá a cada una de las peticiones que contiene.

Cada uno de los métodos del servicio REST se anota con *@GetMapping*, *@PostMapping*, *@PutMapping* o *@DeleteMapping*, especificando a continuación la dirección concreta de la petición. En el caso de las anotaciones *@DeleteMapping* que no devuelven ninguna respuesta, se utiliza *@ResponseStatus(HttpStatus.NO_CONTENT)* para indicar que es una respuesta sin contenido asociado.

Para inyectar los valores de entrada a cada una de las operaciones del controlador REST, se utilizan las anotaciones *@PathVariable* (encargada de inyectar un valor desde la dirección de la petición), *@RequestParam* (inyecta un valor desde un parámetro HTTP) y *@RequestBody* (inyecta los contenidos del cuerpo de la petición).

En el código que se puede ver a continuación se muestran como ejemplo usos de estas anotaciones en la aplicación. Concretamente, estos métodos son los correspondientes a la gestión de categorías perteneciente a *CatalogController*.

```
1  @PostMapping("/categories")
2    public CategoryDto createCategory(@RequestAttribute Long userId,
       @Validated @RequestBody CategoryDto params)
3        throws InstanceNotFoundException, DuplicateInstanceException,
       PermissionException {
4      permissionChecker.checkUserExistsAndIsAdmin(userId);
5
6      return CategoryConversor.toCategoryDto(
7          catalogService.newCategory(params.getName()));
8    }
9
10   @DeleteMapping("/categories/{categoryId}")
11   @ResponseStatus(HttpStatus.NO_CONTENT)
12   public void deleteCategory(@RequestAttribute Long userId,
       @PathVariable Long categoryId)
13       throws InstanceNotFoundException, ProductsExistException,
       PermissionException {
14     permissionChecker.checkUserExistsAndIsAdmin(userId);
15
16     catalogService.deleteCategory(categoryId);
17   }
18
19   @PutMapping("/categories/{categoryId}")
20   public CategoryDto updateCategory(@RequestAttribute Long userId,
       @PathVariable Long categoryId,
21       @RequestBody CategoryDto params) throws
       InstanceNotFoundException, PermissionException {
22     permissionChecker.checkUserExistsAndIsAdmin(userId);
23
24     return CategoryConversor.toCategoryDto(
25         catalogService.modCategory(params.getId(),
       params.getName()));
26   }
27
28   @GetMapping("/categories")
29   public List<CategoryDto> getCategories() {
30     return CategoryConversor.toCategoryDtos(
31         catalogService.findAllCategories());
32   }
```

### Control de acceso y seguridad

En la aplicación se distinguen tres roles correspondientes a los actores definidos en la sección 4.1. Concretamente, la aplicación cuenta con una enumeración de valores implementada como un Enum de Java. Esta enumeración se corresponde con valores asociados en la base de datos, asignando así un rol de la aplicación a cada usuario.

En algunos casos, la aplicación debe distinguir el rol del usuario identificado. Para comprobar que un usuario cliente no pueda realizar operaciones administrativas es necesario tener acceso al identificador del usuario autenticado. Para facilitar estas comprobaciones se utilizan ficheros de configuración de Spring Security, una librería de Spring destinada a la seguridad de aplicaciones.

Para obtener el identificador y el rol de un usuario identificado se utiliza el estándar JSON Web Token (JWT). Este estándar se encarga de afianzar la seguridad de las peticiones mediante un token de identificación. Estos tokens cuentan con tres partes, siendo estas el encabezado, el contenido y la firma. En el contenido o *body* del token cuenta con una fecha de expiración del propio token y la información adjunta al usuario, en este caso su identificador y su rol.

Al tratarse de un token compacto, puede utilizarse fácilmente en las comunicaciones entre el cliente y el servidor. El token generado en la autenticación se almacena en el navegador bajo el *sessionStorage*, es decir, el almacén de datos del navegador correspondiente a una pestaña del mismo. Al estar asociado a una pestaña del navegador, en el momento que la pestaña se cierre, esta información se elimina.

La identificación de los usuarios se realiza mediante un *login* tradicional donde el usuario introduce su nombre de usuario y la contraseña de la cuenta. Para garantizar la seguridad de las cuentas de usuario, la contraseña se encripta en el momento de registro, almacenándose como un String encriptado en la base de datos.

### Internacionalización

El backend debe devolver mensajes de error en caso de que al realizar una operación ocurra una excepción en la aplicación, devolviendo mensajes de feedback en el idioma correspondiente al *locale* del navegador web.

Para implementar esto se utiliza un *bean* de Spring llamado *MessageSource*. Este *bean* se inicializa en el momento de arrancar la aplicación y se encarga de obtener los mensajes de localización que se encuentran como parámetros de un fichero de propiedades. Los mensajes se recuperan cuando es necesario realizando una llamada a *MessageSource* mediante la clave asociada al mensaje y un *locale* específico.

Estos mensajes se envían mediante un DTO llamado *ErrorsDto* que contiene la información del error al lado cliente, que se encarga de mostrarlos cuando sea necesario. Un ejemplo en la aplicación de el uso de estos mensajes es en el control de errores realizado en el backend, en

el momento que una petición REST genera una excepción, esa excepción es reconocida por un método anotado con *@ExceptionHandler(NombreDeLaExcepción.class)*.

Este método se encarga de devolver una respuesta REST conteniendo en el body del mensaje el DTO que contiene el mensaje traducido además de asociar un código de estado HTTP correspondiente con el error, para esto se utilizan las anotaciones de Spring *@ResponseBody* y *@ResponseStatus* respectivamente.

Un ejemplo de uno de estos métodos se puede ver en el siguiente fragmento de código.

```
1                          (...)
2  private final static String INVALID_DATE_RANGE_EXCEPTION_CODE =
       "project.exceptions.InvalidDateRangeException";
3                          (...)
4  @ExceptionHandler(InvalidDateRangeException.class)
5  @ResponseStatus(HttpStatus.BAD_REQUEST)
6  @ResponseBody
7  public ErrorsDto
       handleInvalidDateRangeException(InvalidDateRangeException
       exception, Locale locale) {
8
9      String errorMessage =
       messageSource.getMessage(INVALID_DATE_RANGE_EXCEPTION_CODE, null,
10         INVALID_DATE_RANGE_EXCEPTION_CODE, locale);
11
12     return new ErrorsDto(errorMessage);
13
14   }
```

## 6.2 Implementación del frontend

En esta sección de la memoria se tratan cuestiones de implementación de cada una de las capas del frontend, siendo estas la capa de acceso a servicios web y la interfaz de usuario.

### 6.2.1 Capa de acceso a servicios web

#### Gestión de peticiones HTTP REST

La capa de acceso a servicios web cuenta con un fichero Javascript correspondiente a cada controlador REST definido con anterioridad. Para facilitar la gestión de las peticiones y respuestas se implementa el *wrapper appFetch*.

El fichero *appFetch* utiliza el API Fetch de Javascript [10] para realizar peticiones HTTP mediante la función *fetch*. El *wrapper* cuenta con varios parámetros que puede recibir, siendo estos *path*, *options*, *onSuccess* y *onErrors*.

Concretamente en el parámetro *path* se incluye la dirección de la petición a realizar. El parámetro *options* indica el tipo de petición HTTP (GET, POST, PUT, DELETE...) y permite incluir un objeto al cuerpo de la petición. Por último, los parámetros *onSuccess* y *onErrors* son opcionales y reciben una función que se ejecutará en caso de éxito o error de la petición respectivamente.

### 6.2.2 Interfaz de usuario

#### Estructura de los módulos

Con el objetivo de mantener limpia la implementación realizada, se ha realizado una división por módulos que agrupan los componentes y ficheros necesarios para la funcionalidad de cada módulo declarado en una sección anterior de la memoria (sec. 5.3.2). Para facilitar el mantenimiento de la interfaz de usuario, el código se reparte en una estructura ordenada de ficheros y carpetas donde cada módulo sigue el siguiente patrón organizativo.

- **Components:** Carpeta que contiene los componentes correspondientes a la funcionalidad del módulo. Esta carpeta se ha formado por múltiples subcarpetas para facilitar la organización.
- **actions.js** y **actionTypes.js** contienen la información relevante para Redux con respecto a las acciones que se explican a continuación (sec. 6.2.2).
- **selectors.js:** Define selectores que facilitan el acceso a la información contenida en el estado del módulo.
- **reducer.js:** Gestiona el estado del módulo, manejando los cambios al mismo dependiendo de las acciones realizadas por el usuario.
- **index.js:** Define los componentes exportados por el módulo.

Un ejemplo visual de esta división de contenido de los módulos se puede ver en la figura 6.1. En el caso de este ejemplo se muestran los contenidos del módulo *locations* correspondiente a las funcionalidades del requisito funcional de localizaciones (sec. 4.1).

#### Gestión del estado con Redux

En el proyecto se ha utilizado Redux para manejar la gestión del estado. La ventaja principal que otorga es la modularidad de la librería, permitiendo adaptarse fácilmente a la distribución por módulos indicada en la sección anterior (sec. 6.2.2). Esta modularidad permite separar el propio estado y su lógica de modificación de los componentes individuales, facilitando enormemente el mantenimiento.

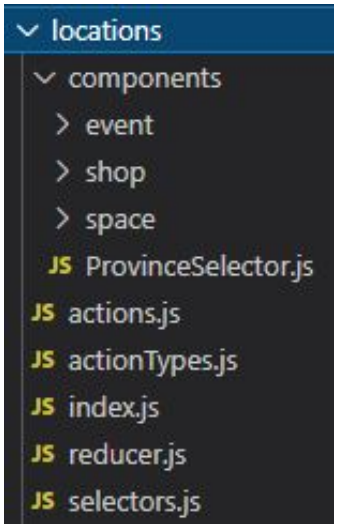


Figura 6.1: Contenidos del módulo locations

El estado se contiene en un almacenamiento o *store* que es modificado cuando ocurren ciertas acciones. Las acciones son objetos generados por la interacción del usuario que contienen la información del tipo de acción a aplicar para permitir que el reductor o *reducer* gestione los cambios relevantes. En cada módulo se definen estos componentes mediante los ficheros *actions.js, actionTypes.js, reducer.js* e *index.js* como se puede ver en la figura anterior (fig. 6.1).

Para gestionar el estado global de la aplicación, se utiliza un reducer raíz. Este reductor contiene los reductores de cada uno de los módulos, combinándolos en uno único mediante la operación *combineReducers* de Redux. En el fragmento de código a continuación se puede ver la definición del reductor raíz de la aplicación, conteniendo los reductores implementados para cada uno de los módulos de la interfaz de usuario.

```
1  import { combineReducers } from 'redux';
2
3  import app from '../modules/app';
4  import users from '../modules/users';
5  import locations from '../modules/locations';
6  import catalog from '../modules/catalog';
7  import retail from '../modules/retail';
8
9  const rootReducer = combineReducers({
10     app: app.reducer,
11     users: users.reducer,
12     locations: locations.reducer,
13     catalog: catalog.reducer,
14     retail: retail.reducer
```

```
});

export default rootReducer;
```

Con el objetivo de permitir acceso a los componentes de la aplicación al estado se utiliza la etiqueta *Provider*. *Provider* se utiliza a un nivel alto de la aplicación, envolviendo al componente *App* que contiene la estructura global de la aplicación. El uso de la etiqueta *Provider* se puede ver en el siguiente fragmento de código.

```
                                (...)
/* Configure store. */
const store = configureStore();

                                (...)

/* Configure i18n. */
const { locale, messages } = initReactIntl();

/* Render application. */
ReactDOM.render(
    <Provider store={store}>
        <IntlProvider locale={locale} messages={messages}>
            <App />
        </IntlProvider>
    </Provider>,
    \textbf{}

                                (...)
```

Para recuperar de forma sencilla la información contenida en un campo del estado se utilizan selectores o *selectors*. Estas son llamadas al estado de la aplicación que devuelven el valor actual de uno de los campos del estado. Para acceder o suscribirse a los cambios de un campo del estado se usa el hook *useSelector* de react-redux [11].

Las acciones ocurren como respuesta a las interacciones del usuario con la interfaz, para lanzarlas se hace uso de otro hook definido por react-redux, en este caso *useDispatch* [11]. El hook facilita una referencia a la función de *dispatch* del store, permitiendo lanzar acciones cuando sean necesarias.

Cuando se lanza una acción, el reductor compara el tipo de la acción (definido con anterioridad en *actionTypes.js*) y ejecuta la lógica asociada para realizar los cambios en el estado. Al realizarse este cambio, todo componente renderizado que esté suscrito mediante un selector al elemento modificado del estado se re-renderiza, reflejando los cambios realizados al estado.

### Control de acceso a componentes

Como se menciona con anterioridad, la aplicación cuenta con varios roles con distintos grados de acceso a la funcionalidad de la aplicación (sec. 4.1). Por tanto, aunque no sustituyen a las comprobaciones realizadas en el backend, se realizan comprobaciones para evitar que usuarios sin permisos puedan acceder mediante interfaz visual a elementos para los que no tienen permisos. Estas comprobaciones ayudan a mejorar la robustez y seguridad de la aplicación.

Para conseguir este objetivo, se definen en el servicio de usuarios un grupo de selectores para obtener el rol del usuario autenticado. Estos selectores permiten comprobar de forma rápida si un usuario es un cliente, un administrador web o un empleado de tienda. En el fragmento de código a continuación se pueden ver los cuatro selectores definidos para controlar accesos.

```
                                (...)
export const isLoggedIn = state =>
    getUser(state) !== null

export const isShopAdmin = state =>
    isLoggedIn(state) && getUser(state).role === 'SHOP_ADMIN' ?
    true : null;

export const isWebAdmin = state =>
    isLoggedIn(state) && getUser(state).role === 'WEB_ADMIN' ? true
    : null;

export const isAdmin = state =>
    isShopAdmin(state) || isWebAdmin(state) ? true : null;
```

Para evitar el acceso y para esconder contenidos o componentes que ciertos tipos de usuarios no están autorizados para visualizar se utiliza el renderizado condicional de React [12]. El renderizado condicional permite renderizar los componentes bajo el cumplimiento de una cláusula if (también representado mediante el operador condicional "&&").

En el código a continuación se muestra un fragmento del fichero *Body.js* donde se utiliza renderizado condicional. En este caso se evita que los usuarios no identificados puedan acceder al componente de visualización de sus pedidos y se impide que cualquier usuario cliente que no tenga capacidades administrativas o sea empleado de una tienda pueda ver los pedidos de otros usuarios.

```
                                (...)
{loggedIn && <Route exact path="/retail/my-orders">
    <FindOrdersResult />
</Route>}
```

```
{isAdmin && <Route exact path="/retail/users-orders/:id">
    <FindUsersOrdersResult />
</Route>}
                                (...)
```

### Internacionalización

Para implementar la internacionalización del frontend se utiliza la librería *React Intl*. Con el objetivo de dar soporte a tres idiomas (gallego, castellano e inglés) se utilizan ficheros de localización que contienen los identificadores y mensajes asociados a cada una de las etiquetas de texto de la aplicación.

Dentro de la estructura del proyecto, los ficheros de localización se encuentran principalmente en la carpeta llamada i18n. En el momento de inicialización de la aplicación web, se recupera el *locale* del navegador y se recuperan los mensajes definidos en los ficheros de localización. Con estos parámetros se inicializa un contexto que envuelve a la aplicación, similar al caso del Provider de Redux, en este caso la etiqueta utilizada se llama *IntlProvider*.

Un ejemplo visual de la etiqueta IntlProvider en la aplicación se encuentra en un fragmento de código anterior (pág. 56). A continuación se pueden ver un par de líneas del fichero de internacionalización a gallego del proyecto *messages_gl.js*.

```
                                (...)
    'project.global.fields.name': 'Nome',
    'project.global.fields.password': 'Contrasinal',
    'project.global.fields.postalAddress': 'Enderezo',
    'project.global.fields.postalCode': 'Código postal',
    'project.global.fields.userName': 'Usuario',
    'project.global.fields.street': 'Rúa',
                                (...)
```

# Capítulo 7

# Pruebas

En este capítulo se explicará el proceso seguido en el desarrollo con respecto a las pruebas realizadas a la aplicación. En el desarrollo de la aplicación se ha seguido una planificación de iteraciones incrementales. Cada iteración se finaliza con la subfase de pruebas.

Se han realizado dos tipos de pruebas, siendo estas pruebas de integración y pruebas funcionales. No se han realizado pruebas unitarias o de aceptación.

El motivo para no realizar las pruebas unitarias individuales es que el código de las entidades y los DAOs es simple y en gran cantidad de las ocasiones generado de forma automática por librerías mencionadas con anterioridad (pág. 32). En la mayoría de los casos, las pruebas de integración sobre los servicios probarán de forma indirecta el código de entidades y DAOs.

## 7.1 Pruebas de integración

Las pruebas de integración son pruebas que abarcan módulos enteros donde elementos individuales se combinan para probarse como un grupo unificado. De esta forma se comprueba que los elementos funcionen correctamente en sus interacciones a través de las interfaces que exponen y que su funcionamiento grupal sea el esperado.

En el desarrollo, estas pruebas se realizan cada vez que se añade un nuevo módulo funcional a la aplicación, permitiendo comprobar y evitar posibles errores en la integración entre ellos.

En el contexto de la aplicación realizada, las pruebas de integración se realizan sobre una base de datos diferente a la utilizada para el desarrollo general, garantizando que esta base de datos no esté afectada por contenidos almacenados con anterioridad y facilitando la limpieza de la misma. Esto se consigue utilizando la anotación *@ActiveProfiles("test")* para activar un perfil de prueba definido en la configuración de la aplicación.

La implementación de las pruebas de integración se realiza usando JUnit. En la aplicación se implementa una clase de pruebas para cada uno de los servicios de la capa de lógica de

negocio, obteniendo un total de cuatro ficheros de prueba. Cada uno de estos tests de servicio se anotan a nivel de clase con *@SpringBootTest*, *@Transactional* y *@ActiveProfiles("test")*.

Cada uno de los métodos de las clases de prueba se anotan utilizando *@Test*. Se han realizado pruebas para casos correctos y para varios casos erróneos de las operaciones de los servicios. El uso de *@Transactional* garantiza que cada uno de los métodos de prueba se ejecuten en una transacción que al encontrarse bajo un entorno de prueba finaliza con un *rollback*, deshaciendo los cambios y evitando contaminación de datos entre pruebas.

Las pruebas siguen una estructura para facilitar su implementación. Cada método de prueba comienza declarando los datos que va a utilizar para la prueba y que no hayan sido definidos a nivel de clase con anterioridad. Después llama a las funciones que van a ser probadas, recopilando los datos obtenidos de dichas operaciones. Por último compara el resultado obtenido con el resultado esperado para la prueba.

En ocasiones se prueban múltiples funcionalidades relacionadas, en este caso los datos obtenidos se comprueban mediante varias aserciones a lo largo de la prueba individual. Un ejemplo de las anotaciones mencionadas y el proceso seguido se puede observar en el código de pruebas *CatalogServiceTest* que se muestra a continuación.

```
@SpringBootTest
@ActiveProfiles("test")
@Transactional
public class CatalogServiceTest {
  @Autowired
  private CatalogService catalogService;

  @Autowired
  private CategoryDao categoryDao;

                                (...)

  private final Long INVALID_ID = -1L;

                                (...)

  /*
   * Category tests
   */

  @Test
  public void testCRUDCategory()
    throws DuplicateInstanceException, InstanceNotFoundException,
    ProductsExistException {
    String testName = "crudTestName";
    Category category = catalogService.newCategory(testName);
```

```
27
28      Category recoveredCategory =
        categoryDao.findByName(testName).get();
29      assertEquals(category, recoveredCategory);
30      assertEquals(recoveredCategory.getName(), testName);
31
32      category = catalogService.modCategory(category.getId(), "New
        name");
33      assertNotEquals(category.getName(), testName);
34
35      catalogService.deleteCategory(category.getId());
36    }
37
38    @Test
39    public void testDeleteNonExistentCategory() {
40      assertThrows(InstanceNotFoundException.class, () ->
        catalogService.deleteCategory(INVALID_ID));
41    }
42                                    (...)
43  }
```

## 7.2 Pruebas sobre la API REST

Para probar el correcto funcionamiento de la API expuesta por la Capa REST (pág. 38), se han realizado comprobaciones utilizando la herramienta Postman. Estas son pruebas de caja negra, es decir, que se realiza una comprobación de las entradas y las salidas del código implementado pero ignorando la estructura concreta del mismo.

Postman es una herramienta para la prueba de APIs REST mediante el envío de peticiones a la misma. En este caso, la herramienta se utiliza con el fin de enviar una petición por cada método expuesto por los controladores de la capa REST.

Entre las opciones de Postman se pueden enviar peticiones HTTP con métodos consistentes a los utilizados en la aplicación (GET, PUT, POST, DELETE...) y adjuntar parámetros o contenidos al cuerpo de la petición. Una ventaja de la herramienta es que permite la utilización del token de identificación, pudiendo comprobar también los accesos de los distintos roles de usuario a la API REST.

Postman permite crear conjuntos de pruebas automatizadas pero en el caso de la aplicación, las pruebas se han realizado de forma individual y manual. Un ejemplo del resultado de una de estas pruebas se puede ver en la figura 7.1.

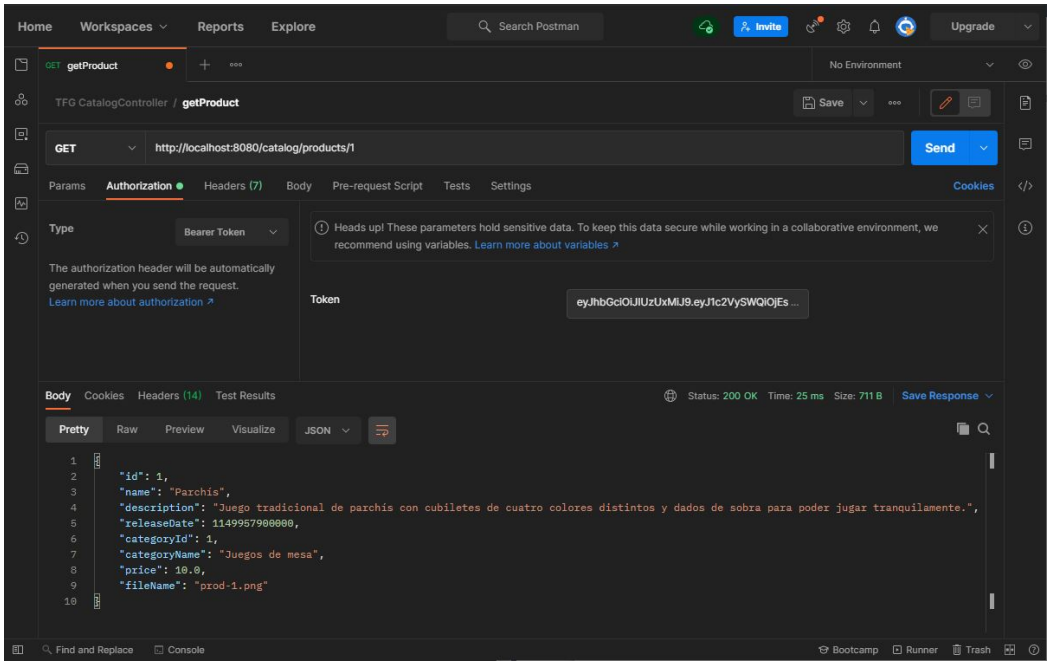


Figura 7.1: Ejemplo de prueba en Postman

## 7.3 Pruebas funcionales

Estas pruebas consisten en la utilización de la interfaz de usuario de forma manual, comprobando que las funcionalidades implementadas se ajustan a los requisitos funcionales establecidos en el capítulo de análisis de la memoria (sec. 4.1). Se han comprobado la mayoría de casos posibles, intentando cubrir casos de error y contemplando mensajes de feedback necesarios.

Para obtener mejores resultados, se han realizado las pruebas múltiples veces, comprobando la consistencia de los resultados de las pruebas consecutivas. Además, se han agrupado elementos funcionales para probarlos de forma conjunta como sería el uso esperado por parte de un usuario final de la aplicación, poniendo atención a que no se produzcan errores derivados de la encadenación de funcionalidades.

Capítulo 8

# Planificación y evaluación de costes

En este capítulo se explican los conceptos de la planificación aplicados al proyecto y como se ajusta esta planificación a la realidad del mismo. Se dará una estimación del tiempo dedicado al proyecto, además de una evaluación de costes siguiendo esa estimación.

## 8.1 Planificación

La planificación es una fase importante de cualquier proyecto software. Con una planificación correcta se puede ver claramente el progreso realizado, se identifican posibles riesgos y se delinean elementos relevantes a priorizar. En el caso del proyecto realizado, la planificación inicial explicada con anterioridad (sec. 4.4) cuenta con 8 iteraciones totales.

La estimación de tiempo de desarrollo del proyecto se puede obtener desde el número de iteraciones planeadas. Asignando un rango de dos semanas para la realización de cada iteración. Cada semana cuenta con cinco días laborables de los que se estima una media de 5 horas de trabajo diario. De estas asignaciones se obtiene la cantidad de horas estimadas de dedicación al proyecto.

**Horas de trabajo estimadas:** 8 iteraciones * 2 semanas * 5 días * 5 horas = 400 horas.

El proyecto comenzó su desarrollo a finales del segundo cuatrimestre del curso 2019/2020. En este momento del proyecto se realizó la definición inicial del mismo, sus objetivos y las posibles iteraciones. Tras varios imprevistos y circunstancias externas al proyecto (principalmente causas personales e inconvenientes técnicos), el avance en las iteraciones se puso en pausa.

Finalmente, el proyecto se retomó a finales de enero de 2021. Siguiendo en la medida de lo posible la planificación realizada e intentando mantenerse dentro de la estimación temporal obtenida. De esta forma, el periodo con mayor carga de dedicación y progreso en el proyecto fueron los cuatro meses siguientes a enero de 2021.

Cabe destacar que aunque no se ha cumplido la planificación inicial en el momento de definición de la misma, si se ha seguido de la forma más cercana posible en el momento de retomar el proyecto, siendo este el periodo de mayor progreso de desarrollo. La planificación inicial estaba planeada para su realización en los meses de mayo, junio, julio y agosto de 2020 según la distribución semanal planeada para las iteraciones (sec. 4.4). La planificación real seguida cumplió el mismo orden de iteraciones planeado en la planificación inicial, pero con un desplazamiento temporal de dichas iteraciones a los meses de febrero, marzo, abril y mayo de 2021.

## 8.2 Evaluación de costes

Para calcular el presupuesto del proyecto es necesario utilizar los datos de planificación obtenidos en la sección anterior de la memoria (sec. 8.1).

Aunque el proyecto fue desarrollado por un único alumno, existen tareas realizadas que son comparables con otros perfiles de recursos humanos del sector. Por tanto, para la estimación se utilizará una separación de las habilidades utilizadas por el alumno como recursos diferentes con distintos objetivos y deberes.

1. **Jefe de proyecto:** Encargado de la definición del proyecto, planificación del mismo y gestión de recursos.

2. **Analista-Diseñador:** Encargado del proceso de solución del problema planteado por el proyecto. Se encarga de las tareas de análisis y diseño.

3. **Programador:** Encargado de las subfases de desarrollo de implementación y pruebas del proyecto.

Para calcular la asignación de horas correspondiente a cada perfil se distribuyen las horas de desarrollo estimadas entre tres tareas principales que pueden realizar los recursos humanos, siendo estas planificación, análisis-diseño e implementación-pruebas.

Debido a que la implementación y las pruebas es la tarea más costosa en tiempo, se le ha asignado un porcentaje estimado del 70% del tiempo total del proyecto. El porcentaje restante se divide en un 5% para la planificación y un 25% para el análisis y diseño. Un resumen visual de esta distribución se puede ver en la tabla 8.1.

Para obtener un resultado correcto debemos tener en cuenta coste de licencias, el coste del equipamiento y posibles impuestos aplicables. En este caso, se ha ignorado el coste de licencias necesarias ya que se ha realizado utilizando librerías y herramientas gratuitas o de código abierto.

| | **Horas** | **Coste por hora** | **Coste** |
|---|---|---|---|
| Jefe de proyecto | 20 | 42€ | 840€ |
| Analista-Diseñador | 100 | 25€ | 2500€ |
| Programador | 280 | 16€ | 4.480€ |
| | | **Coste total:** | 7820€ |

Tabla 8.1: Asignación de costes y horas a los recursos

En el caso del coste de equipamiento, el proyecto se ha realizado en un equipo informático personal que no ha requerido de ningún tipo de ampliación o mejora. Este equipo cuenta con otros usos personales además del proyecto y se asume que ha amortizado su vida útil. Por tanto, el equipo usado no supone un cambio relevante ni significativo en el presupuesto final obtenido y se ha ignorado el coste asociado.

En la tabla que se muestra a continuación (tab. 8.2) se puede ver el resultado final de la estimación de presupuesto, teniendo en cuenta el Impuesto sobre el Valor Añadido (IVA) de 21% aplicable en España en 2021.

| | **Coste asociado** |
|---|---|
| Recursos humanos | 7820€ |
| Licencias y equipamiento | 0€ |
| Impuestos aplicables | *IVA* (21%) |
| **Coste total:** | 9462,20€ |

Tabla 8.2: Coste total estimado del proyecto

# Capítulo 9
# Conclusión

## 9.1 Conclusiones

### 9.1.1 Sobre los objetivos iniciales

Para obtener una imagen global del estado del proyecto y las soluciones aplicadas es necesario comparar los resultados obtenidos del desarrollo del proyecto con los objetivos planteados anteriormente en la introducción de la memoria (sec. 1.3).

1. En cuanto a la gestión de usuarios, se ha conseguido cubrir la autenticación y registro de usuarios, el registro de nuevos empleados de tienda, la visualización de los usuarios y también la posibilidad de gestionar el estado de pago o cancelación de los encargos de los clientes por parte de los administradores y empleados. Para los clientes se posibilita la opción de cancelar alguna de sus reservas o pedidos mientras se encuentran pendientes de confirmación de pago.

2. Sobre el objetivo de gestión del almacén se ha conseguido implementar el control CRUD de productos, ya sea añadiendo productos, modificándolos o eliminando aquellos introducidos por error del catálogo. De cada producto pueden existir múltiples instancias de cualquiera de los tres tipos principales definidos en los objetivos, siendo estos venta, venta de segunda mano y producto para alquiler. Estas instancias individuales también cuentan con operaciones básicas de control de las mismas y se pueden visualizar cómodamente tanto como cliente a la hora de añadirlas al carrito o como empleado o administrador web para ver el stock disponible de un producto concreto.

3. Sobre el control de alquileres de espacios se ha conseguido mostrar los espacios individuales de cada tienda en la aplicación, posibilitando el control de la capacidad máxima y coste por día de cada uno por parte de los administradores web y empleados de tienda, pudiendo comprobar la disponibilidad para días o periodos de tiempo concretos. Es-

tos espacios pueden ser comprobados también por los clientes que tienen la opción de alquilarlos para fechas específicas según consideren.

4. Sobre el objetivo de ampliación sencilla del número de espacios o tiendas de la empresa. Se ha conseguido implementar control administrativo de las mismas, permitiendo la creación y modificación de tiendas y espacios. También se dispone de eliminación de tiendas y espacios insertados por error o que se consideren innecesarios y no hayan sido utilizados en la aplicación. Como se ha mencionado en capítulos anteriores (sec. 4.1), los permisos de eliminación y creación de nuevas tiendas son exclusivas para los administradores web.

5. Sobre la gestión de eventos se ha conseguido implementar un control básico de los mismos con operaciones CRUD para los administradores web y empleados de tienda. Los usuarios pueden buscar de forma cómoda entre los eventos disponibles en la plataforma mediante la búsqueda avanzada de eventos o pueden ver los eventos próximos de tiendas concretas desde la página de detalles de una tienda en la que estén interesados.

6. En cuanto al objetivo de expandir y afianzar conocimientos en los frameworks, lenguajes y herramientas utilizadas, se ha notado un aumento en la velocidad de desarrollo y una adquisición de soltura con las herramientas más profunda que con la que se contaba al principio del proyecto. Aunque la mayoría eran tecnologías conocidas para el alumno, existen aspectos de desarrollo que se han comprendido mejor gracias a la realización del proyecto.

7. El último objetivo consistía en obtener una versión funcional de la aplicación en cada iteración. La aplicación fue ampliada progresivamente, intentando completar lo máximo posible de cada funcionalidad esperada para el periodo de tiempo establecido por la iteración. En ocasiones, algunas funcionalidades implementadas tuvieron que ser revisadas y ampliadas debido a otros cambios en la aplicación.

También se ha conseguido realizar un diseño adaptable o *responsive* que permite de forma cómoda el uso de la aplicación en un entorno que no sea un equipo informático de escritorio, facilitando la accesibilidad de la plataforma desarrollada.

Se considera que el resultado final cumple los objetivos planteados inicialmente. La aplicación resultante es funcional, pero al tratarse de una aplicación desarrollada dentro de un entorno académico no cuenta con todos los aspectos que deberían considerarse en una aplicación finalizada para un contexto comercial. Algunos de los aspectos que podría ser interesante mejorar o ampliar se comentarán en la sección siguiente sobre líneas futuras (sec. 9.1.1).

### 9.1.2 Líneas futuras

Existen múltiples opciones de ampliación para la aplicación desarrollada, algunas de estas funcionalidades fueron consideradas en la planificación original del proyecto pero fueron descartadas debido al tamaño actual de la aplicación o porque no se consideraron críticas para el proyecto.

#### Estilos y temas web

Una posible línea de mejora podría ser la ampliación de la aplicación para mostrar distintos aspectos o temas personalizables. Actualmente muchas aplicaciones web o nativas de escritorio cuentan con dos temas principales, siendo estos tema día y tema noche. Estos temas fueron planteados para incluirse a la aplicación en un punto intermedio del desarrollo pero se decidió posponerlo como una línea futura de trabajo.

#### Mejoras a la gestión de usuarios

En primer lugar, podría resultar interesante para los administradores web o empleados de tiendas controlar de forma restrictiva a los usuarios, permitiendo restringir su acceso a la aplicación en caso de que existan perfiles con intenciones maliciosas o conflictivas, además de poder realizar bloqueos permanentes o temporales.

Para asegurarse de que los usuarios cuentan con toda la información posible sobre su tienda más cercana se planteó la posibilidad de añadir una tienda como tienda favorita. Esta funcionalidad permitiría al usuario elegir el local en el que está más interesado y recibir notificaciones cuando se añaden nuevos eventos o se producen cambios en la tienda en su correo electrónico o en un panel de notificaciones de la aplicación.

Podría también resultar interesante para los usuarios permitir la valoración de los productos que hayan comprado o alquilado en alguna ocasión, asociando incluso un comentario con una pequeña crítica del producto. Estas valoraciones se podrían mostrar en la información del producto con una opción para leer las críticas asociadas.

Otra funcionalidad interesante para los usuarios podría ser una lista de amigos dentro de la aplicación, permitiendo hacer reservas conjuntas entre grupos de usuarios que acuden juntos a los eventos de la plataforma.

#### Mejoras sobre eventos

En cuanto a la gestión de eventos, la implementación actual es bastante simple y básica. A continuación se destacan un par de formas de ampliar este apartado de la aplicación.

Permitir a los usuarios mostrar interés en eventos concretos mediante *likes*. Esto permitiría a los administradores comprobar de forma sencilla qué eventos interesan más a los clientes

facilitando la toma de decisiones de la empresa.

Podría ser interesante poder asociar eventos con espacios concretos, permitiendo reservar los propios espacios para controlar la asistencia en caso de que haya múltiples eventos simultáneos en un mismo establecimiento.

También podría ser interesante añadir una lista de productos asociados al evento, en algunos casos como las presentaciones de producto podría ser útil para el usuario saber fácilmente sin tener que leer la descripción completa del evento si está relacionado con sus intereses.

#### Noticias de la empresa

Este concepto es, en esencia, similar a los eventos implementados en la aplicación. Estas noticias podrían resultar interesantes como mensajes que se muestran cómodamente en un formato blog. También podría considerarse una integración de estas noticias con plataformas de redes sociales, de las cuales muchas empresas del ámbito de tiendas de ocio hacen uso frecuentemente para sus promociones.

Estas noticias podrían ser publicadas por los administradores y se mostrarían en la página principal de la aplicación, permitiendo su visualización de forma cómoda. Además, si se vinculan con tiendas concretas, los usuarios que con anterioridad hayan marcado la tienda relacionada con la noticia como tienda favorita podrían recibir notificaciones o avisos.

#### Visualización en calendario

Para mejorar la comodidad de los alquileres, pedidos y reservas de espacio realizadas por los usuarios podría ser de interés implementar una vista en calendario de las mismas. Esta vista permitiría al usuario comprobar rápidamente las fechas futuras sin tener que buscar en el historial de pedidos, alquileres o reservas de espacio.

#### Traducciones ampliadas

Con el fin de globalizar el uso de la aplicación, puede resultar interesante para los clientes tener acceso a los detalles de las entidades almacenadas en la base de datos como productos o tiendas localizado a su idioma nativo. Para esto podría implementarse una forma de traducción automática que se encargue de dicho problema o una forma de almacenar la información de los productos para distintos idiomas que soporta la aplicación.

#### Soporte para tarjetas de crédito

Por comodidad para cualquier aplicación de compra-venta es necesario contar con una integración con pasarelas para compra online mediante tarjeta de crédito. Como la intención de la aplicación es crear una plataforma con fines educativos se consideró inicialmente soportar

una pasarela de pago con tarjetas de crédito, pero el principal enfoque en el desarrollo ha sido cumplir con los aspectos funcionales de gestión de los elementos de la empresa, por tanto no se ha considerado como una función crítica para el proyecto, quedando pospuesto el soporte para tarjetas de crédito como una posible línea futura de ampliación.

## 9.2 Lecciones aprendidas

A continuación se explican varias lecciones aprendidas o recordadas a lo largo del proyecto.

### Sobre la simplicidad

Durante el desarrollo de la aplicación se han encontrado varios desafíos a la hora de solucionar los problemas planteados en la definición inicial de la misma. Algunos casos concretos como la visualización de tiendas junto a la disponibilidad de los espacios de las mismas probaron ser algo más complejas de lo que parecían en la concepción del proyecto.

La principal herramienta a la hora de abarcar este problema han sido los mockups y bocetos del aspecto de la interfaz realizados durante el desarrollo. Estos dibujos y simulaciones sencillas han probado ser importantes para la finalización del proyecto planeado.

La división en componentes simples de ciertos casos complejos, como el mencionado anteriormente, ha supuesto una reducción en el tiempo de desarrollo inesperada y realmente valiosa. En resumen, la simplicidad apremia.

### Sobre la reutilización de código

La reutilización de código ha sido un recurso indispensable en la realización del proyecto. Gracias a la estructura por componentes, estos se pueden reutilizar de forma sencilla y ayudan a mantener el *look and feel* de la aplicación. Este concepto ha ayudado enormemente con la finalización ágil de varias funcionalidades esperadas en la aplicación, ahorrando tiempo de desarrollo.

### Sobre el trabajo individual

A lo largo del grado existen múltiples asignaturas en las que se deben realizar trabajos en equipo con unas condiciones similares a las establecidas en este proyecto. Por lo general se realizan en grupos de dos o más personas, facilitando el desarrollo enormemente al poder repartir distintos objetivos a realizar entre los alumnos. En este caso al tratarse de un desarrollo individual es necesario conocer todo detalle de funcionamiento del proyecto para su éxito.

### Sobre la planificación

La planificación ha sido de gran relevancia para el desarrollo del proyecto, bastante más de lo esperado inicialmente. La planificación inicial sirvió como guía del trabajo realizado y pendiente.

Aunque no se siguió la planificación para el periodo de tiempo planteado inicialmente como se comentó en el capítulo anterior (cap. 8), la división en iteraciones y la organización de las funcionalidades a implementar intentó seguirse en el momento de retomar el proyecto, resultando vital para la realización del mismo.

### Sobre posibles nuevas funcionalidades

Durante las distintas fases de análisis, diseño e implementación de la aplicación, la implementación de ciertos elementos funcionales de la aplicación daba lugar a la idea de nuevas posibles funcionalidades para la aplicación.

En un caso real, puede resultar común encontrarse con cambios de parecer por parte del cliente o equipo de desarrollo sobre las funcionalidades definidas inicialmente para un proyecto. Esto suele ser consecuencia de cambios en el entorno del proyecto o posibles mejoras a realizar que se salen del ámbito definido para la aplicación.

En el caso del proyecto realizado ha sido importante centrarse en las funcionalidades definidas inicialmente para conseguir un resultado final, de otra forma este proyecto podría alargarse de forma prácticamente indefinida.

# Apéndices

Apéndice A

# Manual de usuario

En este anexo se muestran vistas varias de la aplicación para ilustrar de una forma visual el posible uso por parte de un usuario final.

## A.1 Autenticación y registro

En esta sección se muestra como acceder y registrarse en la aplicación. Desde la página de inicio de la aplicación mostrada en la figura A.1 se puede acceder a la autenticación de usuario.

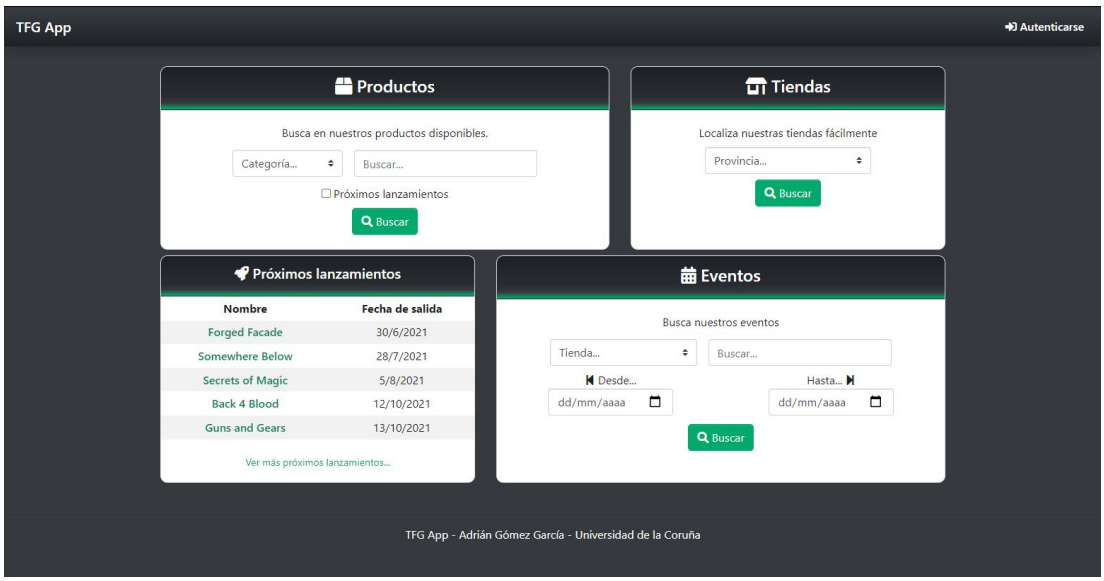


Figura A.1: Pantalla de inicio

En caso de que el usuario sea un nuevo cliente de la aplicación puede registrarse cómodamente haciendo click en el enlace ”Registrarse” de la pantalla de autenticación de usuario (fig. A.2), este enlace lleva al usuario a la pantalla de registro de nuevo cliente (fig. A.3).

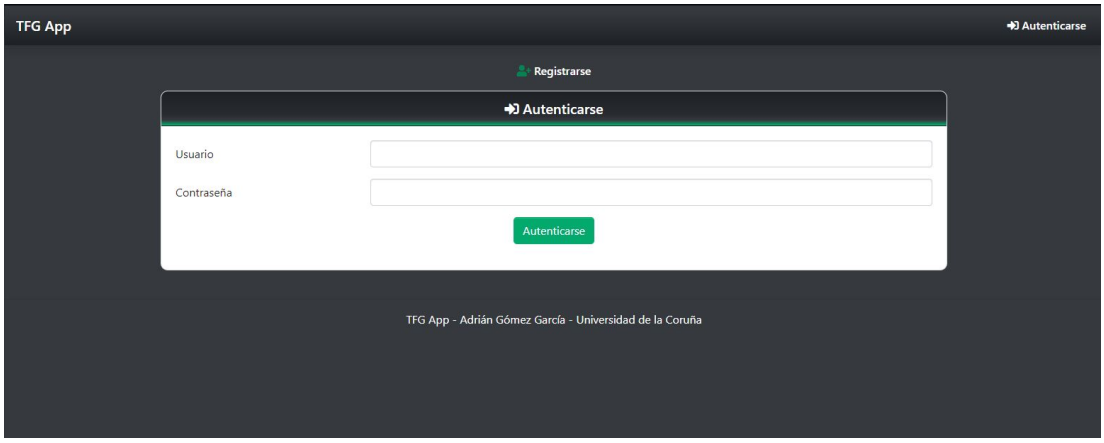


Figura A.2: Pantalla de autenticación de usuario

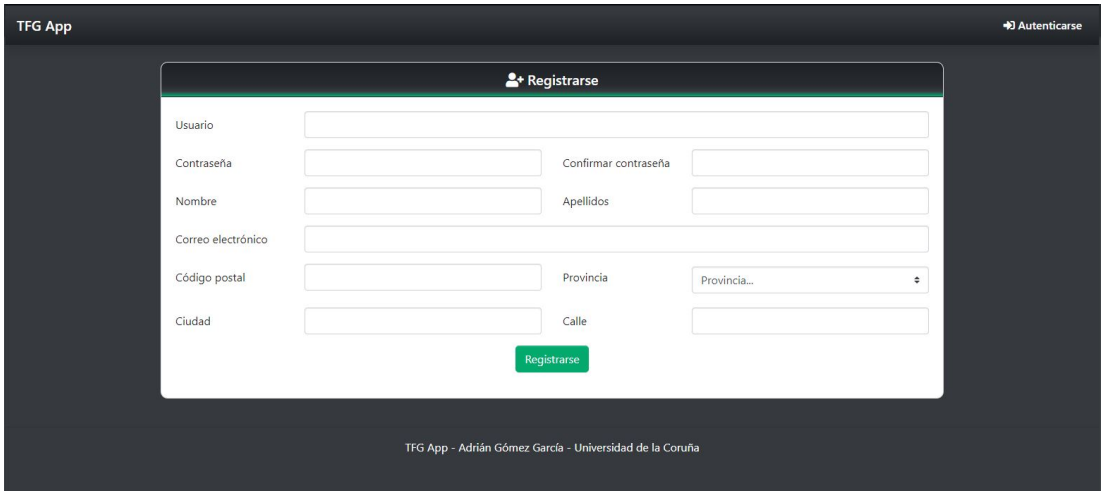


Figura A.3: Registro de nuevo cliente

Una vez registrado el usuario rellenando los campos del formulario, se le autentica en la aplicación, permitiendo acceso a funcionalidades de usuario para las que no tiene acceso un usuario sin identificar. Estas se pueden ver en la sección siguiente sobre funcionalidades de usuario (sec. A.2).

## A.2 Funcionalidades de usuario

En cuanto el cliente haya creado una cuenta mediante el registro de usuarios o acceda a su cuenta a través de la pantalla de autenticación, el cliente gana acceso a varias funcionalidades como su carrito de la compra y sus historiales de pedidos y reservas.

### A.2.1 Funcionalidades básicas y perfil de usuario

En la esquina superior derecha de la pantalla se puede ver el contador de productos presentes en el carrito de la compra y un menú de opciones de usuario (fig. A.4).

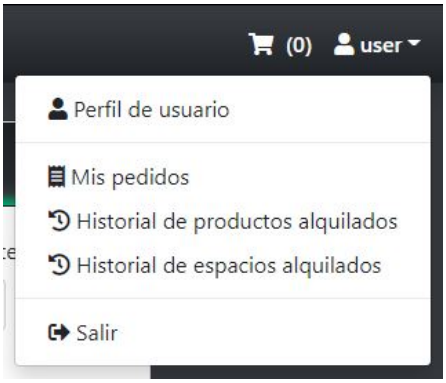


Figura A.4: Menú de opciones de cliente

El cliente autenticado puede visualizar su perfil haciendo click en la opción ”Perfil de usuario” del menú de opciones. Además, puede comprobar los pedidos de productos realizados, el historial de productos alquilados y también el historial de espacios alquilados.

Si el usuario accede a su perfil, la información se muestra como en la figura A.5. Desde esta vista, un usuario puede modificar su perfil haciendo click en el botón de ”Actualizar perfil”. Además, puede cambiar su contraseña desde el botón adyacente.

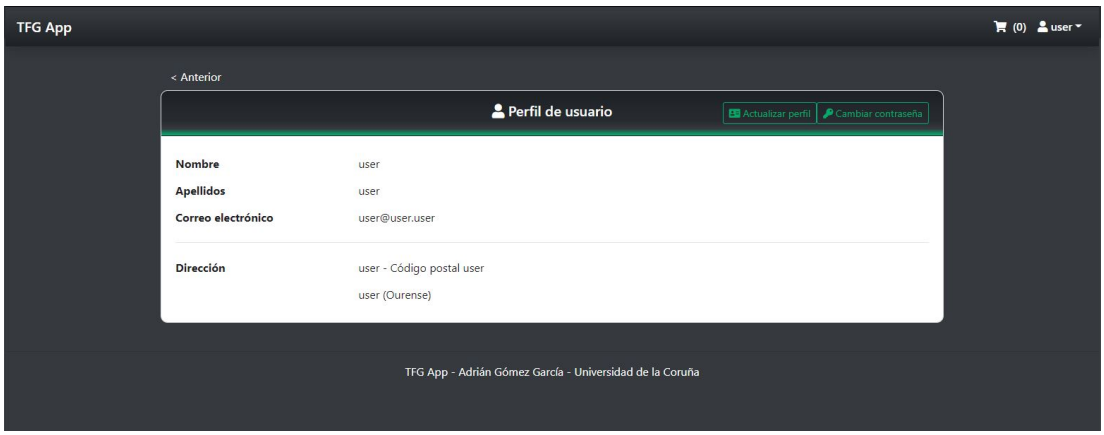


Figura A.5: Mi perfil de usuario

El formulario de actualización de perfil es similar al formulario de registro mostrado con anterioridad en la sección A.1, excepto que no cuenta con campos para actualizar la contraseña. En la aplicación se utiliza otra vista para esta función que se puede ver en la figura A.6


Figura A.6: Cambio de contraseña

### A.2.2 Pedido de productos

A continuación se mostrará cómo realizar un pedido de productos paso a paso. Para empezar, desde la pantalla de inicio de la aplicación accedemos a la sección de búsqueda de productos e introducimos los parámetros que queremos para realizar la búsqueda. En el ejemplo de la figura A.7, se buscan juegos de mesa sin filtro por palabras clave.


Figura A.7: Formulario de búsqueda

En el momento que se pulsa el botón de búsqueda, la aplicación muestra una lista paginada de productos que cumplan las condiciones de la búsqueda. Un ejemplo de la lista resultante se puede ver en la figura A.8.

Cuando se hace click sobre la imagen en miniatura del producto o sobre el enlace con el nombre del mismo, se accede a los detalles del producto.

En la página de detalles se puede ver la información desglosada del producto, permitiendo ver inicialmente una lista de instancias disponibles para añadir al carrito de la compra como se puede ver en la figura A.9.

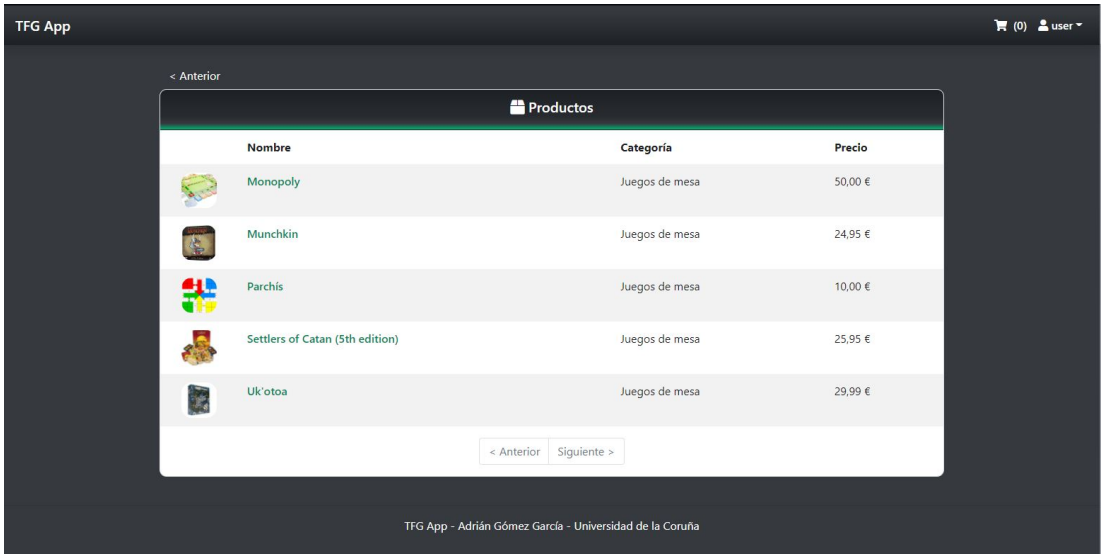


Figura A.8: Resultado de la búsqueda de productos

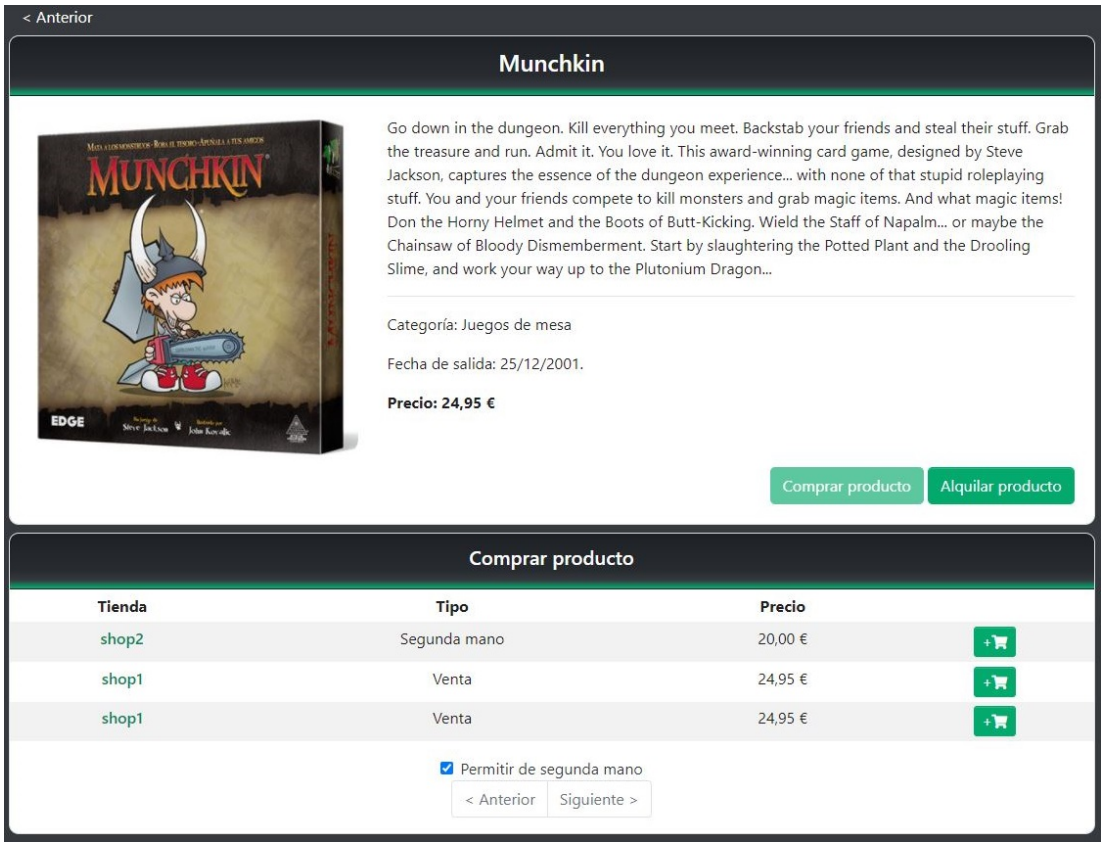


Figura A.9: Detalle de un producto, compra

Cuando el cliente ve una oferta de interés, puede añadirla al carrito mediante el botón verde disponible en la lista de opciones de compra. Al hacer click en este botón, el elemento desaparece de la lista y se incrementa el número situado arriba a la derecha de la pantalla con el icono del carrito de la compra.

Si se pulsa el icono del carrito, se muestra la lista de productos presentes en el carro junto con un botón para confirmar la compra. En cualquier momento previo a la confirmación, el cliente puede eliminar elementos concretos del carrito en los que ya no esté interesado pulsando el botón rojo con el icono de una papelera (fig. A.10).

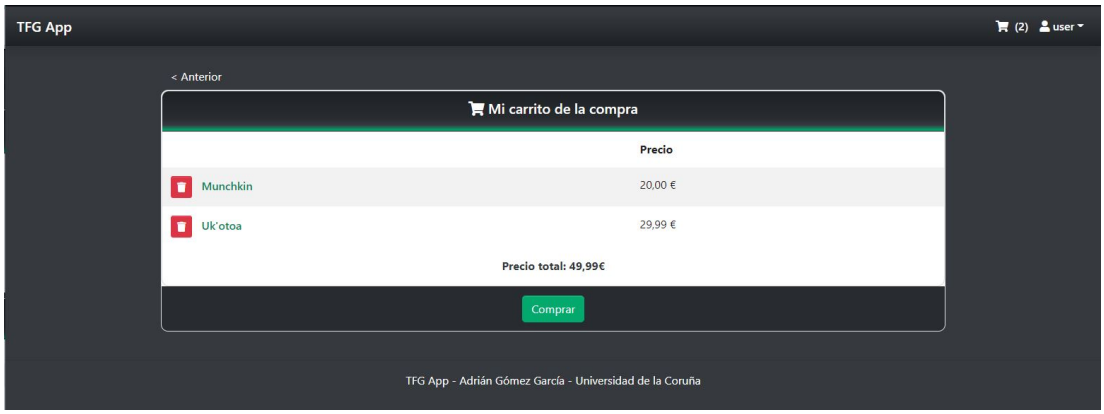


Figura A.10: Elementos del carrito de la compra

Una vez confirmada la compra se muestra un formulario de confirmación. En esta página se selecciona la tienda de recogida del pedido además de mostrar el precio total del pedido a realizar. En cuanto se confirma este formulario, se añade el pedido confirmado a la lista de pedidos del usuario y se muestra un mensaje de confirmación con un enlace al mismo y el identificador del pedido. Esto se puede ver en las figuras A.11 y A.12.

### A.2.3 Alquiler de productos

En el caso de que el usuario prefiera alquilar producto para disfrutarlo durante un tiempo limitado, se puede pulsar en el botón de ”Alquilar producto” en la página de detalles de un producto. Al pulsar este botón, se mostrará un nuevo formulario para seleccionar el rango de fechas de duración del alquiler y la tienda en la que se quiere realizar la reserva (fig. A.13).

En cuanto el usuario ha introducido las fechas y la tienda, si existen copias de dicho producto disponibles para alquiler se muestra un formulario de confirmación similar al mostrado anteriormente para el caso de una compra. Este formulario se puede ver en la figura A.14.

Una vez el cliente confirma el alquiler, se muestra un mensaje de confirmación similar al mostrado anteriormente para el caso de los pedidos. Este mensaje contiene un enlace que permite al usuario ver los detalles de su alquiler de producto.

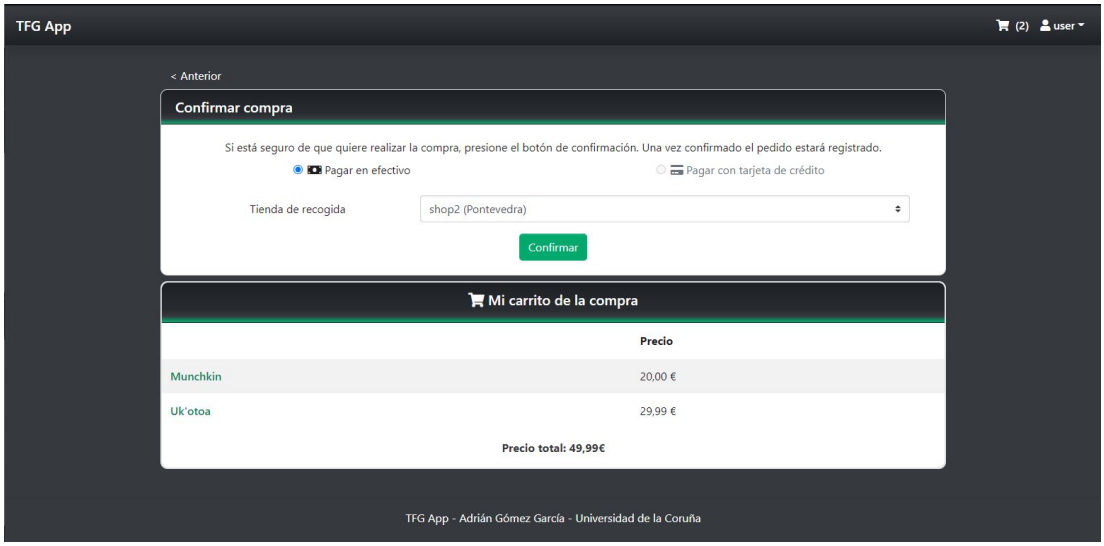


Figura A.11: Vista de confirmación de pedido


Figura A.12: Mensaje de confirmación de pedido

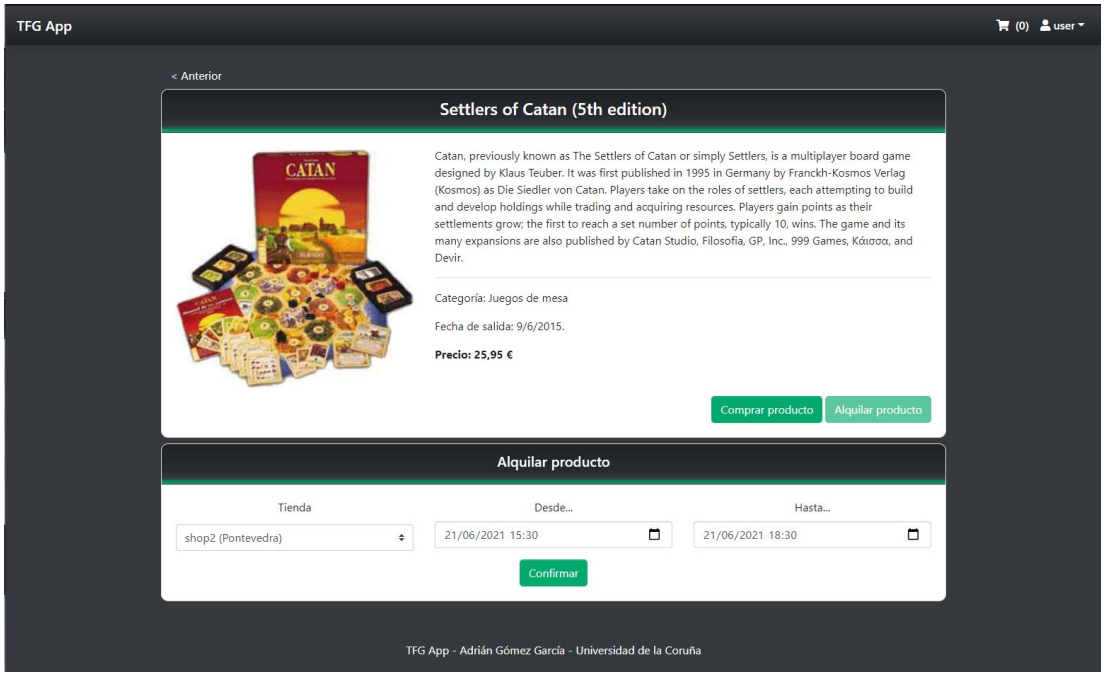


Figura A.13: Detalle de un producto, alquiler

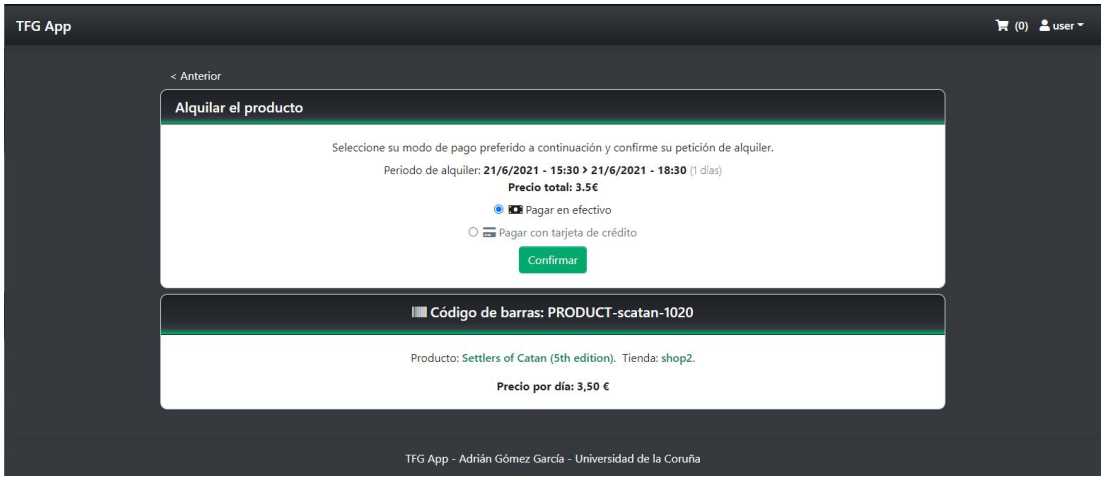


Figura A.14: Alquilar un producto

### A.2.4 Alquiler de espacios

Los clientes cuentan también con la opción de reservar espacios de una tienda. Para esto es necesario entrar a los detalles de la tienda en la que el cliente está interesado. Empezando desde la pantalla de inicio mostrada anteriormente (fig. A.1), los usuarios pueden ver las tiendas disponibles utilizando la búsqueda de tiendas (fig. A.15).


Figura A.15: Búsqueda de tiendas

Para el ejemplo se buscará que tiendas están disponibles en la provincia de Pontevedra, pero el usuario puede buscar en cualquier otra provincia disponible en la aplicación además de realizar una búsqueda de todas las tiendas existentes en cualquier provincia si no selecciona una concreta.

Cuando el usuario pulsa el botón de búsqueda, la aplicación muestra una lista de las tiendas que cumplan los criterios esperados. Un ejemplo de esta lista de tiendas se puede ver en la figura A.16


Figura A.16: Lista de tiendas

Pulsando en el enlace con el nombre de la tienda concreta se accede a los detalles del establecimiento. En esta vista se pueden distinguir los eventos de la semana actual y un selector de espacios disponibles en la tienda (fig. A.17).

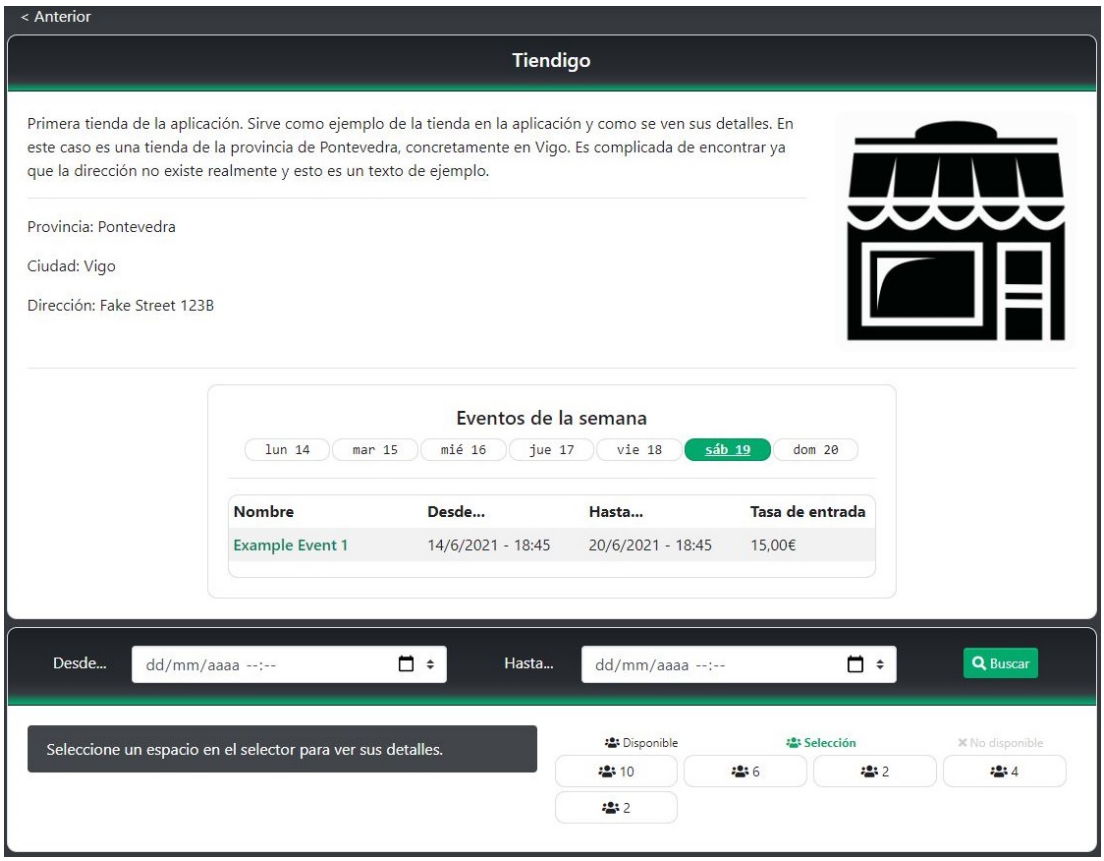


Figura A.17: Detalles de una tienda

Una vez seleccionado un rango de fechas, se puede ver la disponibilidad de los espacios indicada por el icono de cada espacio. El número indica la capacidad máxima del espacio. Si las fechas seleccionadas son correctas y se encuentran en el futuro, el botón llamado ”Reservar

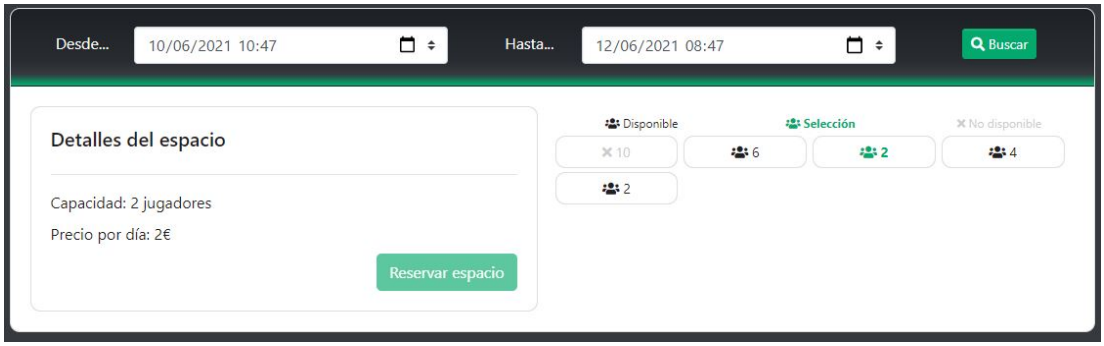


Figura A.18: Selector de espacios

espacio" se activa, permitiendo reservar el espacio.

Si el botón de "Reservar espacio" se pulsa, la aplicación muestra un formulario de confirmación de alquiler similar al utilizado con anterioridad en los alquileres de productos (fig. A.14), que tras ser confirmado muestra un mensaje de confirmación con un enlace como el usado para confirmar el carrito de la compra (fig. A.12).

En caso de intentar reservar un espacio que no se encuentra disponible para las fechas seleccionadas porque ya cuenta con una reserva, se mostrará un mensaje de error como en el ejemplo de la figura A.19.

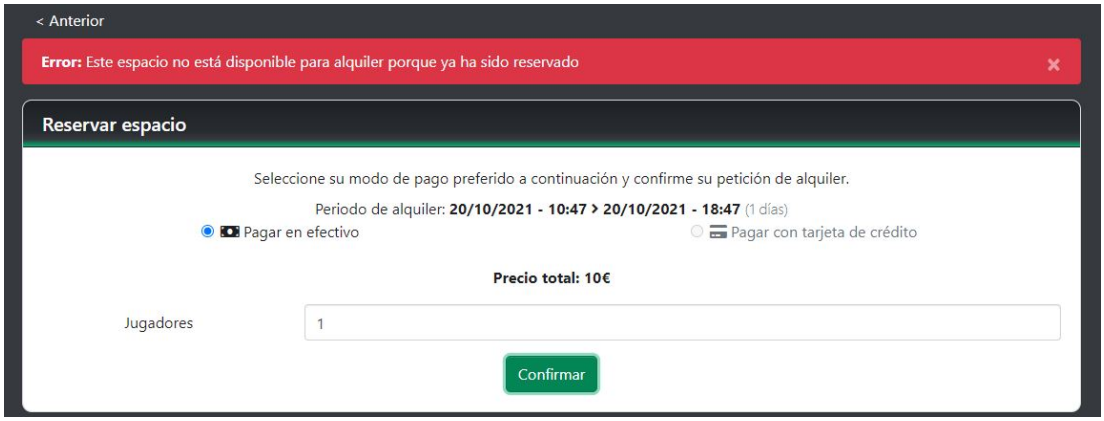


Figura A.19: Ejemplo de error en la reserva de espacio

### A.2.5 Visualización de históricos

La aplicación cuenta con formas de ver los pedidos, alquileres de productos y reservas de espacio desde el menú mostrado con anterioridad (fig. A.4). Al hacer click sobre "Mis pedidos" se puede ver una lista como la mostrada en la figura A.20.

Hacer click en "Historial de productos alquilados" o "Historial de espacios alquilados" lleva a una vista como las mostradas en las figuras A.21 y A.22.

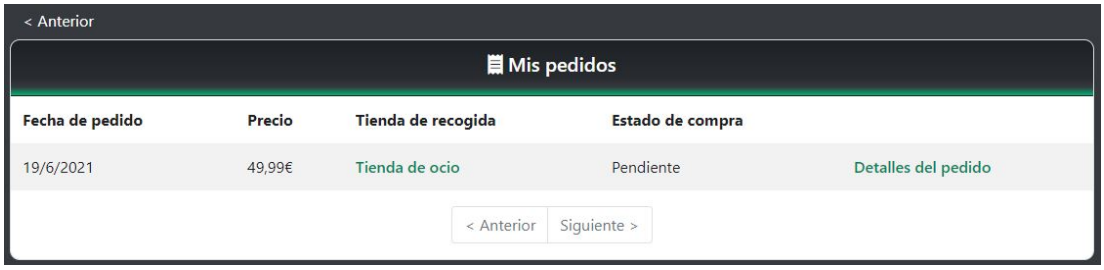


Figura A.20: Pedidos de un usuario

Ambas vistas tienen la opción de filtrar los resultados para mostrar solo los que tengan una fecha futura y no hayan sido cancelados.


Figura A.21: Historial de reservas de producto


Figura A.22: Historial de reservas de espacio

Desde estas listas se puede acceder a los detalles de cada pedido o reserva del usuario, pudiendo ver los detalles de los mismos y en caso de que se encuentre en estado pendiente de pago pueda cancelarlo. Se muestra como ejemplo los detalles de un pedido de productos en la figura A.23.

En caso de querer cancelar cualquier tipo de pedido, se mostrará una ventana modal de confirmación al usuario. Un ejemplo de esta ventana se puede ver en la figura A.24.

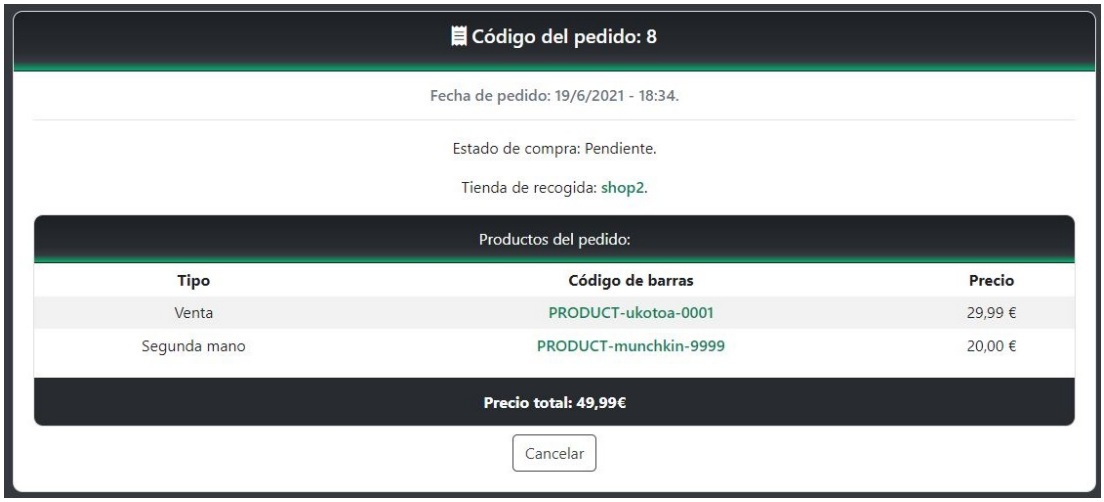


Figura A.23: Detalle de un pedido

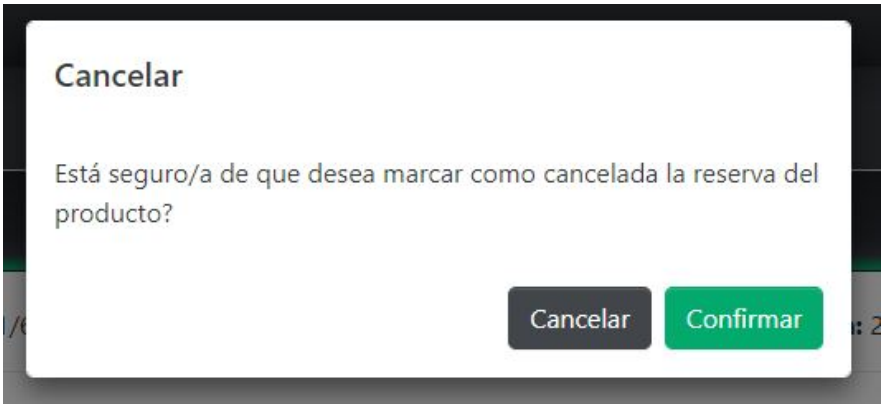


Figura A.24: Confirmación de cancelación

## A.3 Funcionalidades de administración

A continuación se muestra el uso de las funcionalidades administrativas y de gestión de la plataforma. Por simpleza, en esta sección se explican las funcionalidades de administrador web y de los empleados de forma conjunta.

### A.3.1 Panel de control

Los administradores web y empleados de tienda cuentan con paneles de control diferentes. Desde estos paneles tienen acceso rápido a las funcionalidades de gestión de la aplicación. Para acceder al panel de control, se utiliza el menú de opciones situado en el header de la aplicación que se puede ver en la figura A.25.

Tras pulsar el enlace ”Panel de control” la aplicación muestra las opciones que están dis-

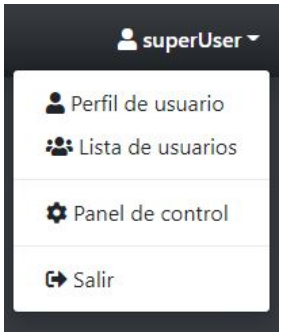


Figura A.25: Menú de un administrador o empleado

ponibles para el usuario. En el caso de un administrador web, el panel tendrá el aspecto de la figura A.26. Para un empleado de tienda, el panel cuenta con las opciones que se ven en la figura A.27.


Figura A.26: Panel de control de un administrador web

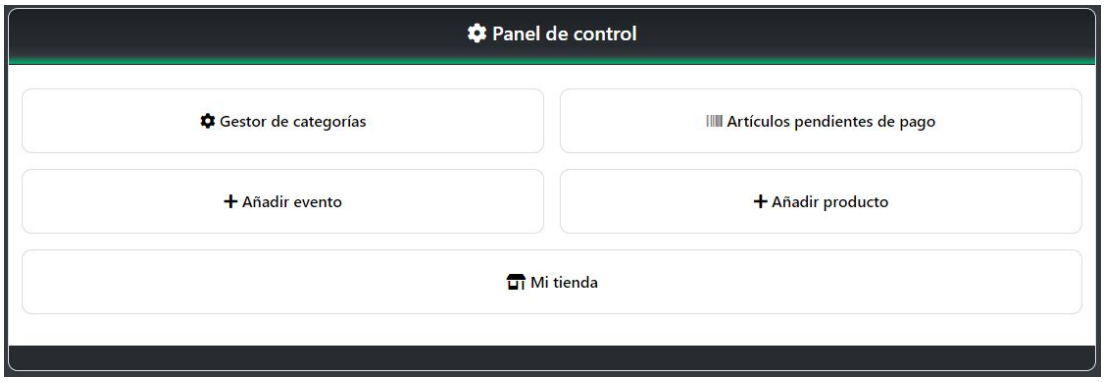


Figura A.27: Panel de control de un empleado de tienda

### A.3.2 Gestión de categorías

Para facilitar la organización del catálogo de la aplicación, los administradores web y empleados de tienda tienen acceso al gestor de categorías que se puede ver en la figura A.28.

Para editar una categoría se pulsa el botón de la columna actualizar, que permite alterar el nombre de la categoría sin salir del gestor de categorías.

En el caso de que se quiera eliminar una categoría sería necesario únicamente pulsar el botón rojo con el icono de la papelera. Cuando se intenta eliminar una categoría se mostrará un mensaje de confirmación para evitar borrados por error como se puede ver en la figura A.29.

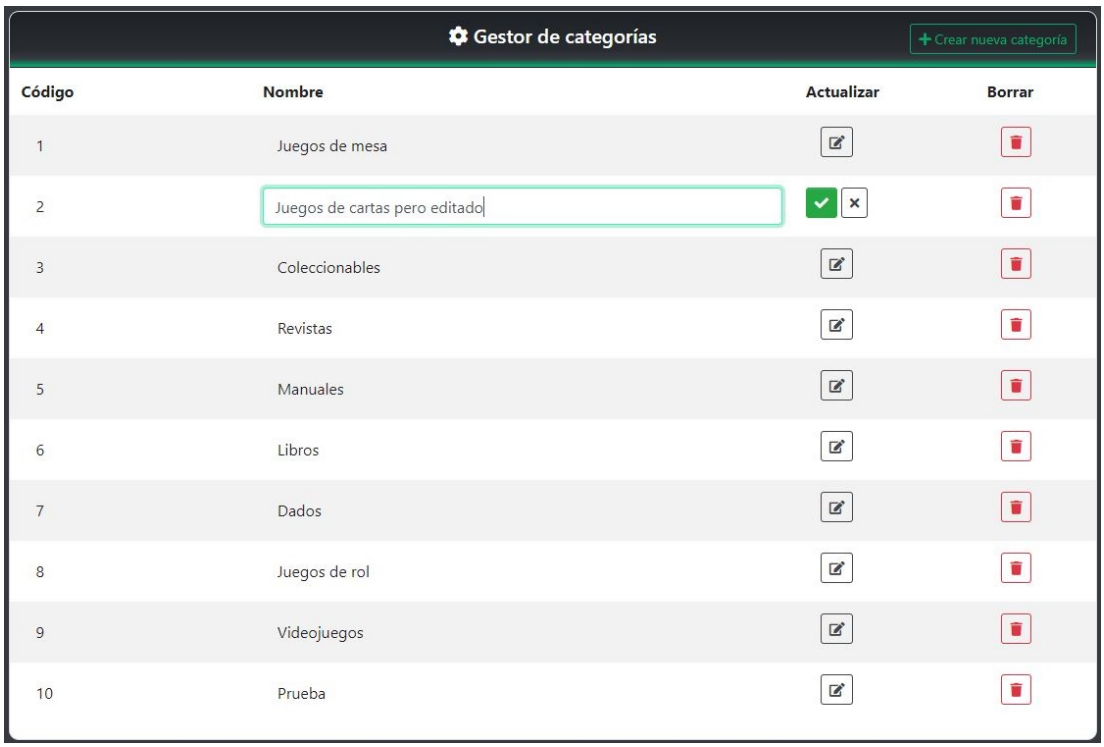


Figura A.28: Gestor de categorías

A la hora de crear una nueva categoría, se utiliza el botón ”Crear nueva categoría” disponible en el gestor encima de la columna de borrado. Este botón lleva a una página que permite insertar nuevas categorías mediante un formulario de un único campo, siendo este el nombre de la categoría a añadir.

### A.3.3 Revisión de pedidos pendientes de pago

Para comprobar los pedidos, alquileres de productos y reservas de espacio que se encuentran en un estado pendiente de confirmación, los administradores cuentan con un panel donde


Figura A.29: Mensaje de confirmación al borrar categoría

pueden comprobar estos tres aspectos de forma rápida.

Para acceder a esta funcionalidad, es necesario pulsar el botón de ”Artículos pendientes de pago” del panel de control. En cuanto se pulsa, la aplicación muestra las listas en el formato que se puede ver en la figura A.30

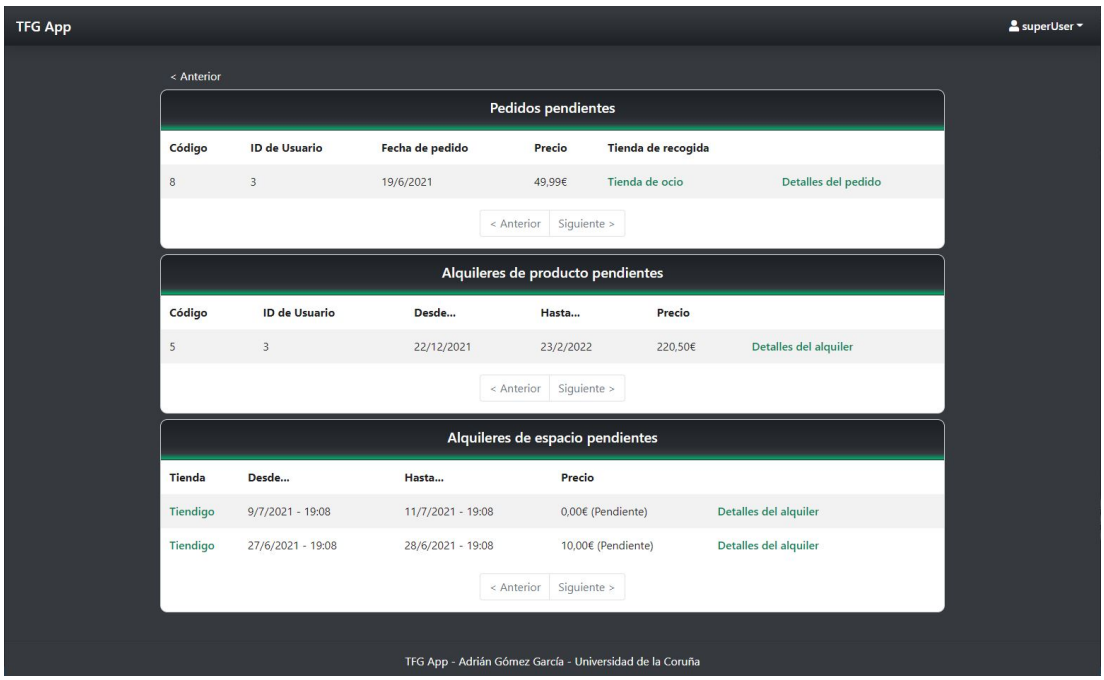


Figura A.30: Artículos pendientes de pago

Desde este panel se puede acceder a la información de cualquier pedido pendiente de cualquier usuario, permitiendo marcarlos como pagados o cancelados según sea necesario. Un ejemplo de cómo se ven los detalles de un pedido desde un perfil de administrador se puede ver en la figura A.31.

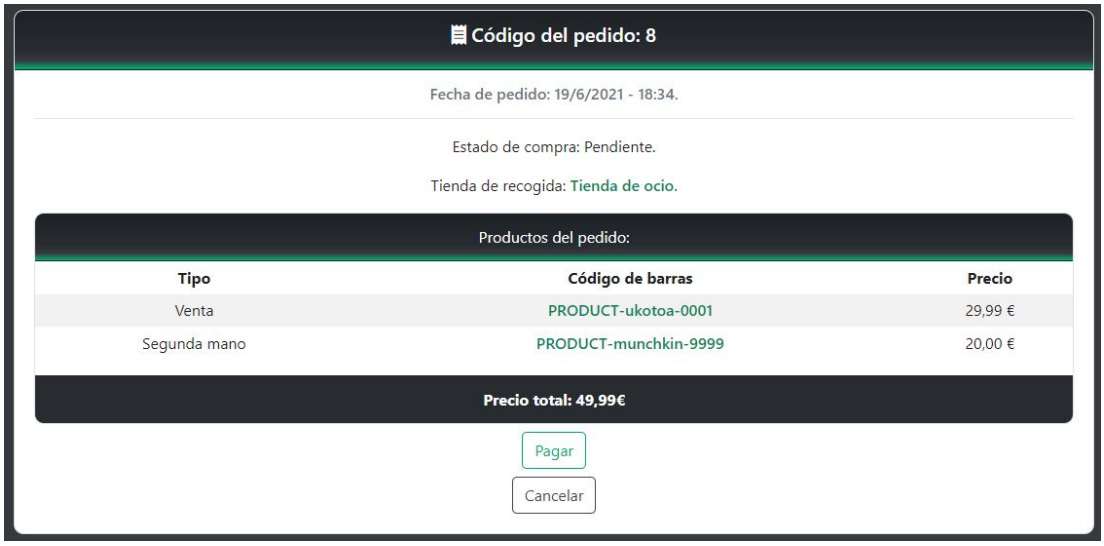


Figura A.31: Ver pedido como administrador

Pulsar el botón de cancelar muestra un mensaje como el de la figura A.24. Pulsar el botón de pagar muestra un mensaje de confirmación con la misma forma pero con el mensaje que se puede ver en la figura A.32

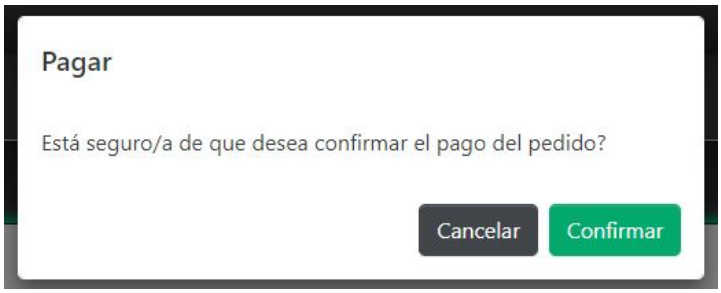


Figura A.32: Confirmación de pago

### A.3.4 Añadir nuevos elementos

Los administradores web y empleados de tienda pueden añadir nuevos elementos a la aplicación. En el caso de los administradores web pueden añadir nuevos productos, nuevos eventos y nuevas tiendas desde el panel de control. Como empleado de tienda se pueden añadir nuevos eventos a su tienda o nuevos productos al catálogo.

Como estas introducciones de elementos son formularios similares, a modo de ejemplo se puede ver el formulario para añadir productos en la figura A.33.


Figura A.33: Añadir producto al catálogo

### A.3.5 Registrar empleado de tienda/Mi tienda

Esta funcionalidad exclusiva de administrador web permite añadir nuevos empleados. Se trata de un registro de usuarios similar al mostrado con anterioridad (fig. A.3), siendo la principal diferencia la necesidad de añadir una tienda a la que este empleado estará asociado como se ve en la figura A.34.


Figura A.34: Registrar nuevo empleado de tienda

En el caso de estar autenticado como un empleado de tienda, el botón para registrar un nuevo empleado de tienda se reemplaza por ”Mi tienda”. En este caso el botón redirigirá al usuario a la página de detalles de la tienda a la que está asociado en lugar de permitir el registro de empleados de tienda.

### A.3.6 Lista de usuarios

Desde el menú de usuario en la parte superior derecha de la aplicación se puede acceder a la lista de usuarios usando el enlace ”Lista de usuarios”. Una vez pulsado se muestra la lista paginada de todos los usuarios de la aplicación, permitiendo filtrarlos mediante una búsqueda por su nombre de usuario como se muestra en la figura A.35.

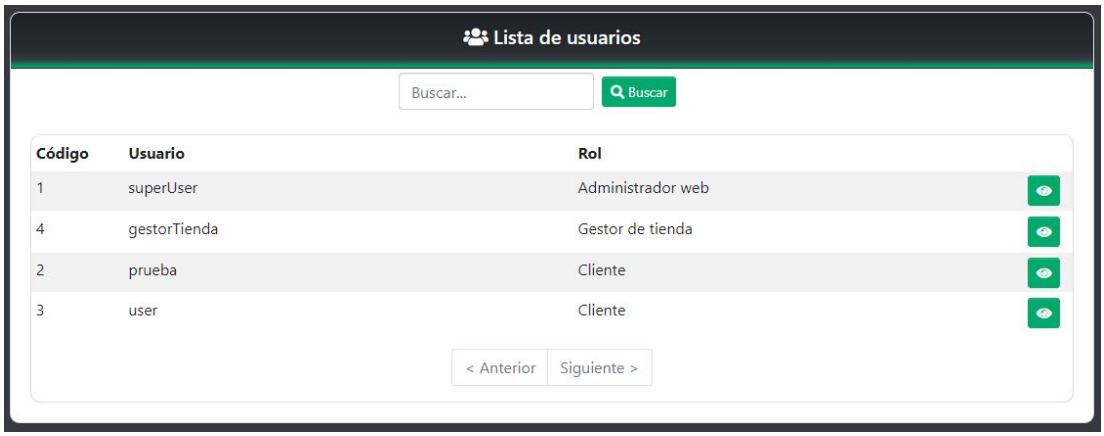


Figura A.35: Lista de usuarios

Utilizando el botón con el icono del ojo se puede acceder a los detalles de un perfil de usuario concreto, permitiendo revisar los pedidos, alquileres de producto y reservas de espacio de ese usuario concreto de forma sencilla. Por brevedad, se muestra los detalles de un perfil de usuario visto por un gestor de la aplicación en la figura A.36

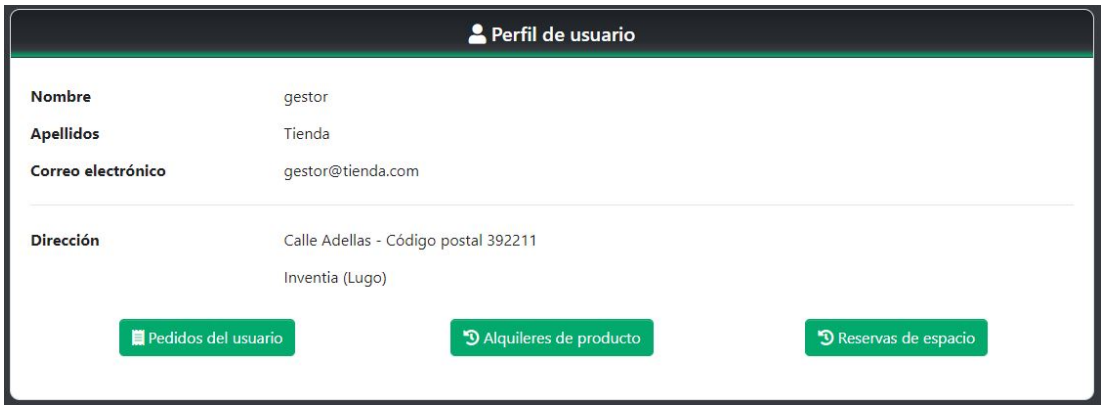


Figura A.36: Perfil de usuario con histórico

### A.3.7 Gestión de productos

Para facilitar la gestión de los productos disponibles en la aplicación, desde la ventana de detalles de un producto se muestran varias funcionalidades que no están disponibles para clientes. Estas funcionalidades como editar la información, la imagen o controlar las copias individuales de un producto se pueden ver en las figuras A.37 y A.38.


Figura A.37: Ver stock de un producto

En el caso de que se quiera borrar un producto del catálogo, basta con pulsar el botón de ”Eliminar producto” a la derecha del título del producto a eliminar, entonces se abrirá una ventana de confirmación como las mostradas con anterioridad en otros casos de gestión (fig. A.29).

En casos donde el producto cuente con stock o copias registradas en la aplicación, no se permitirá borrar el producto, mostrando un error como el que se ve en la figura A.39.

Para añadir nuevas copias disponibles o stock para un producto, es tan sencillo como pulsar el botón ”Añadir copias disponibles” que abrirá un formulario donde introducir los datos de la instancia como el código de barras, tienda en la que se encuentra, tipo del producto y su precio. Un ejemplo exitoso se puede ver en la figura A.40.

La aplicación también permite editar cada instancia individualmente y eliminar copias concretas cuando no cuenten con ningún tipo de reserva o interacción. Esto se hace de forma similar a la manera de realizarlo sobre un producto del catálogo, por tanto se omiten en este

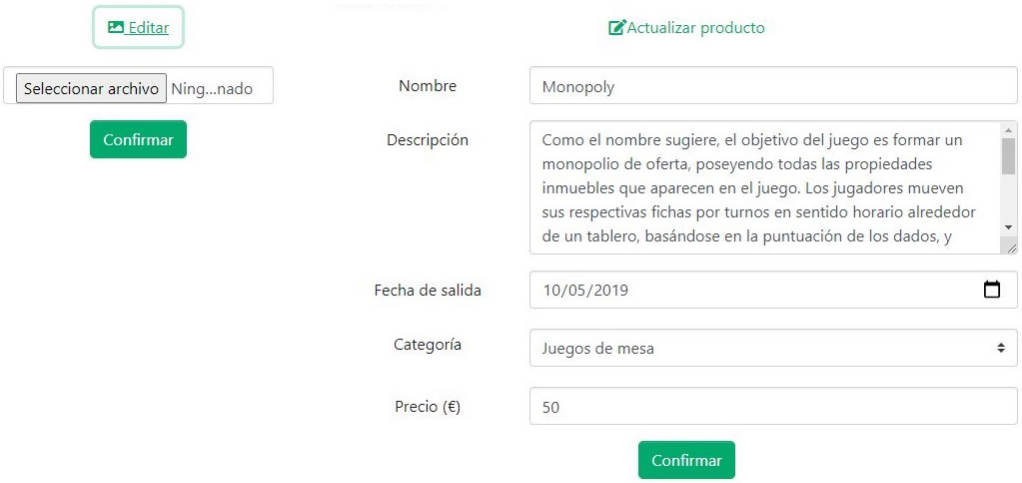


Figura A.38: Editar un producto o su imagen

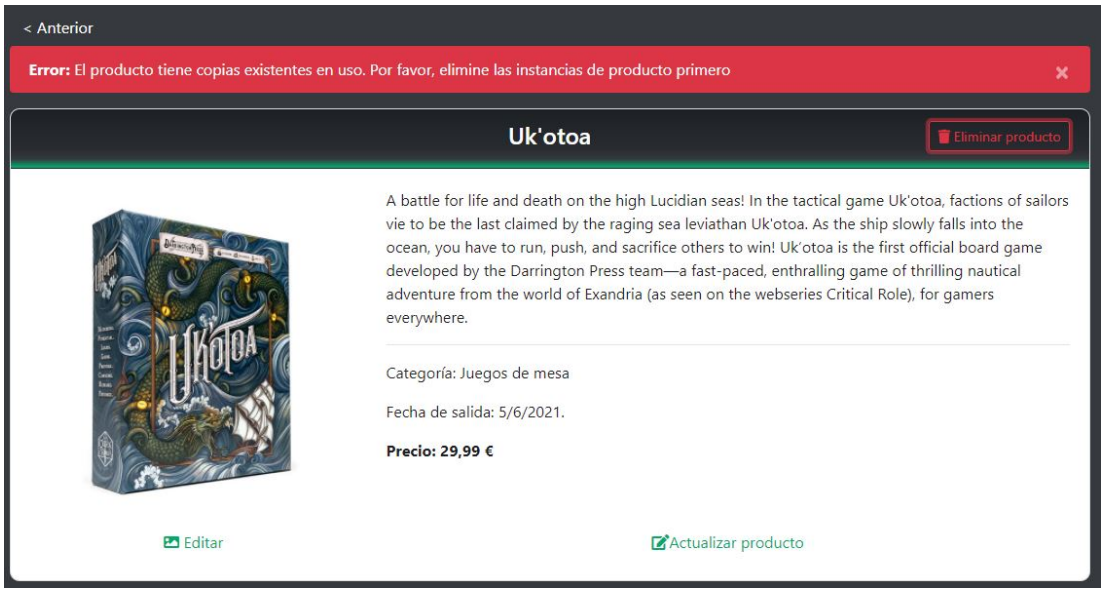


Figura A.39: Error al eliminar un producto

manual por brevedad.

### A.3.8 Gestión de tiendas

De una forma similar a la explicada en la sección anterior sobre los productos (sec. A.3.7) se pueden manejar las tiendas individuales. Un ejemplo de como se ve la página de detalles de una tienda por parte de un gestor de la aplicación se puede ver en la figura A.41.

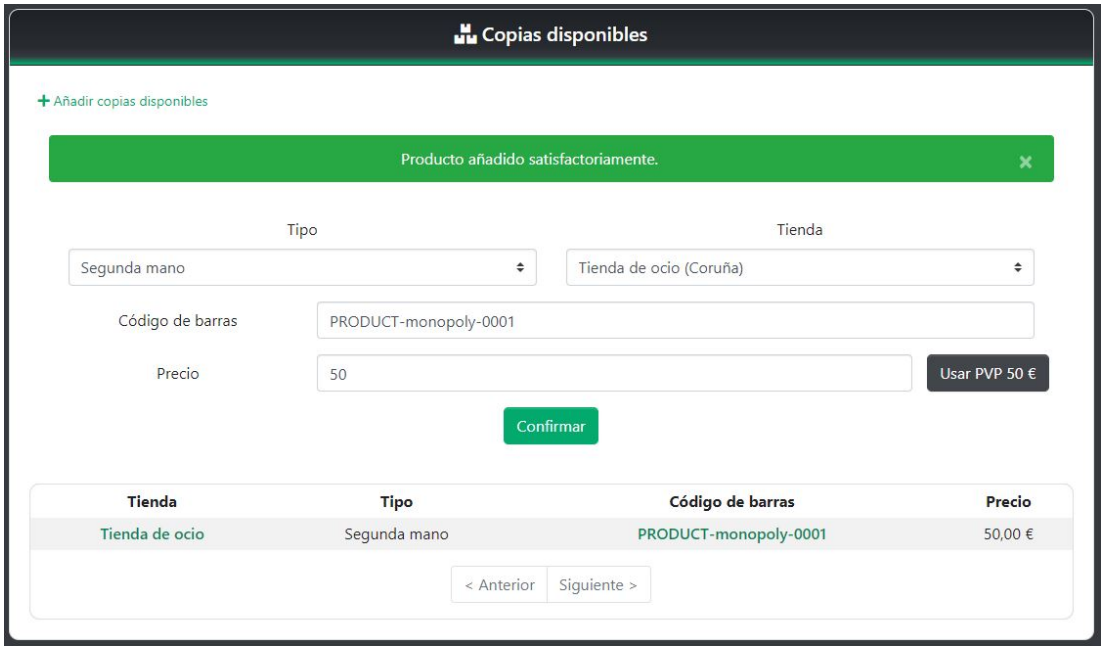


Figura A.40: Añadir stock a un producto


Figura A.41: Editar detalles de tienda

### A.3.9 Gestión de eventos

De una forma muy similar a la gestión de tiendas y productos, es posible para los administradores eliminar y modificar tiendas siguiendo unas reglas similares a las explicadas en las secciones anteriores. Unos ejemplos visuales se pueden ver en las figuras A.42, A.43 y A.44.

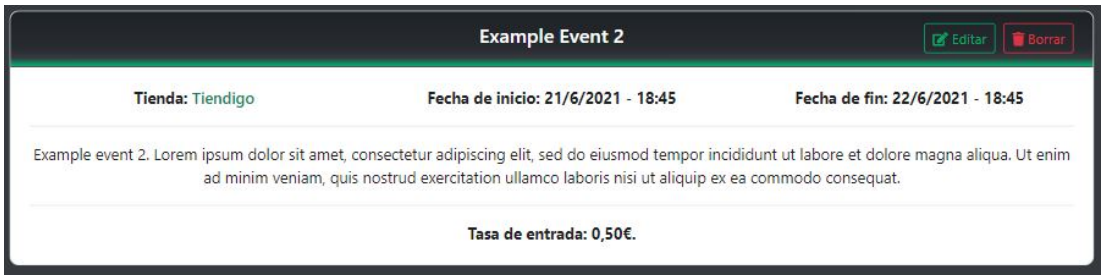


Figura A.42: Detalles de un evento


Figura A.43: Editar detalles de evento


Figura A.44: Confirmación de eliminación de evento

# Lista de acrónimos

**API** Application Programming Interfaces. 7

**CRUD** Create, Read, Update and Destroy. 15, 16, 33

**CSS** Cascading Style Sheets. 6

**DAO** Data Access Object. 7, 32, 33, 37

**DOM** Document Object Model. 8

**DTO** Data Transfer Object. 38, 39

**HTML** HyperText Markup Language. 6

**HTTP** Hypertext Transfer Protocol. 39

**IDE** Integrated Development Environment. 6, 7

**IVA** Impuesto sobre el Valor Añadido. 65

**JPA** Java Persistence API. 7

**JPQL** Java Persistence Query Language. 5

**JWT** JSON Web Token. 52

**POM** Project Object Model. 6

**REST** Representational State Transfer. 7, 8, 38, 39

**Sass** Syntactically Awesome Stylesheets. 6

**SPA** Single-Page Application. 29

**SQL** Structured Query Language. 5

**XP** Extreme Programming. 11

# Bibliografía


[1] "Pypl popularity of programming language." [En línea]. Disponible en: https://pypl.github.io/PYPL.html

[2] React, "Presentando jsx." [En línea]. Disponible en: https://es.reactjs.org/docs/introducing-jsx.html

[3] "Sass home page." [En línea]. Disponible en: https://sass-lang.com/

[4] J. Holmgren, "Yarn 1 vs yarn 2 vs npm," 2020. [En línea]. Disponible en: https://shift.infinite.red/yarn-1-vs-yarn-2-vs-npm-a69ccf0229cd

[5] React, "Presentando hooks," 2018. [En línea]. Disponible en: https://es.reactjs.org/docs/hooks-intro.html

[6] "React redux home page." [En línea]. Disponible en: https://react-redux.js.org/

[7] "Moment.js project status." [En línea]. Disponible en: https://momentjs.com/docs/#/-project-status/

[8] "Moment.js on npm." [En línea]. Disponible en: https://www.npmjs.com/package/moment

[9] "Los doce principios del manifiesto ágil," 2001. [En línea]. Disponible en: https://agilemanifesto.org/principles.html

[10] "Fetch api, referencia de la api web." [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/Fetch_API

[11] "React-redux hooks api." [En línea]. Disponible en: https://react-redux.js.org/api/hooks

[12] React, "Renderizado condicional de react." [En línea]. Disponible en: https://es.reactjs.org/docs/conditional-rendering.html